ENCICLOPEDIA DEL BASIC

LeG

SPECTRUM 2

Estructura de la programación



ceac

SPECTRUM 2

Estructura de la programación

ENCICLOPEDIA
DEL
BASIC

SPECTRUM 2

Estructura de la programación



Perú, 164 - 08020 Barcelona - España

Primera edición: Enero 1987

ISBN 84-329-7712-8 (Rústica)
ISBN 84-329-7717-9 (Rústica especial)
ISBN 84-329-7722-5 (Cartoné)
ISBN 84-329-7729-2 (Cartoné especial)
ISBN 84-329-7726-8 (Cartoné, Obra completa)
ISBN 84-329-7727-6 (Cartoné, Obra completa especial)

Depósito Legal: B-44038 - 1986

Impreso por
GERSA, Industria Gráfica
Tambor del Bruc, 6
08970 Sant Joan Despí (Barcelona)

Printed in Spain
Impreso en España

Contenido

Cómo estudiar en esta Enciclopedia

Al comenzar cada Capítulo encontrará un *esquema de contenido*. La finalidad de este esquema es doble:

— Por una parte, le ofrece una visión panorámica de todos los temas que va a estudiar en ese capítulo.

— Por otra, si posteriormente debe repasar algún punto determinado, le facilitará el poder localizarlo.

Leálos despacio para tener esta visión panorámica.

Después del esquema de contenido, cada capítulo comienza definiendo sus objetivos. De esta manera usted sabrá desde el principio lo que aprenderá en él. También le servirá como referencia para saber si el objetivo marcado ha sido alcanzado por usted.

A lo largo de los capítulos y al final de los mismos encontrará unos *resúmenes*, seguidos de unos *ejercicios de autocomprobación*. Su finalidad es hacer un alto en el camino y recapitular lo estudiado desde la última parada. Al mismo tiempo, los ejercicios de autocomprobación le servirán para que usted mismo compruebe si ha asimilado los conceptos estudiados y si está en disposición de continuar adelante o, por el contrario, si es necesario volver a repasar algo antes de seguir. La solución a los ejercicios de autocomprobación la encontrará al final de la primera parte del volumen.

Le recomendamos también que, cuando deba interrumpir su estudio, lo haga siempre al final de un capítulo o al terminar de resolver alguno de los grupos de ejercicios de autocomprobación que aparecen a lo largo de las lecciones.

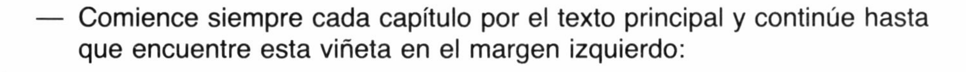
Al hablar de la composición de la Enciclopedia, le dijimos que cada volumen se componía de dos partes. La del texto principal o estudio del BASIC estándar, y la de «Práctica con el microordenador», que viene a ser un complemento de la anterior. Ahora que va a comenzar a estudiarlo comprenderá enseguida la razón de esta comparación.

Cuando se aprende un idioma es frecuente que uno se encuentre con la sorpresa de que en determinadas regiones aquello que uno aprendió a decirlo de una manera se diga de otra. Con el lenguaje de programación BASIC pasa lo mismo. Cada fabricante introduce en el uso de su ordenador un dialecto distinto, que normalmente coincidirá en su mayor parte con el

BASIC estándar, pero tendrá suficientes características especiales como para que el que comienza se encuentre ante su ordenador sin saber qué hacer en determinados momentos.

Por eso, como nuestra intención es que aprenda BASIC y que lo practique con *su microordenador* hemos utilizado la Enciclopedia de modo que usted no se encuentre perdido en ningún momento. Así, al estudiar la lección del BASIC estándar es como si dijéramos: *Esto se dice así normalmente en lenguaje BASIC*. Sin embargo, cuando haya diferencias, añadiremos en el capítulo «Prácticas con el microordenador»: *Ojo, que esto mismo en su microordenador se dice de esta forma*. De este modo, podrá dialogar tranquilamente con su microordenador sin desagradables sorpresas y traducir al lenguaje que entiende su microordenador un programa que en BASIC estándar pueda estar escrito de forma un poco diferente.

Concretamente, en el estudio de las dos partes debe proceder del modo siguiente:

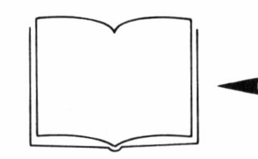


— Comience siempre cada capítulo por el texto principal y continúe hasta que encuentre esta viñeta en el margen izquierdo:



Verá que en la pantalla de la viñeta aparecen unos números. Concretamente 1.1. Esto indica que debe dejar en este momento el capítulo que está estudiando, y pasar al apartado 1.1. de «Prácticas con el microordenador».

— Continúe ahora el estudio de «Prácticas con el microordenador» hasta que encuentre esta otra viñeta:



En ella se le remite de nuevo al capítulo que dejó anteriormente, para continuar donde lo interrumpió. También aquí se le indica el sitio exacto donde debe continuar. En todo caso, le recomendamos que deje siempre una señal en la página donde interrumpe el estudio. Así tendrá siempre la página localizada.

Observará el lector de esta «Enciclopedia del BASIC» que la numeración de capítulos es correlativa, iniciándose en el Tomo I de la misma y finalizando en el Tomo V y último.

Hemos creído conveniente tal continuidad, en primer lugar por el carácter secuencial de la Enciclopedia, al ser los temas que la constituyen correlativos entre sí y formando un todo inseparable; y en segundo lugar porque estimamos que ello facilita la *rápida localización de cualquier tema*, lo que, al tratarse de una obra de consulta frecuente, supone una indudable ventaja para el lector.

Aclaración importante

Observará el lector de esta «Enciclopedia del BASIC» que la numeración de capítulos es correlativa, iniciándose en el Tomo I de la misma y finalizando en el Tomo V y último.

Hemos creído conveniente tal continuidad, en primer lugar por el carácter secuencial de la Enciclopedia, al ser los temas que la constituyen correlativos entre sí y formando un todo inseparable; y en segundo lugar porque estimamos que ello facilita la *rápida localización de cualquier tema,* lo que, al tratarse de una obra de consulta frecuente, supone una indudable ventaja para el lector.

Parte I
BASIC

Capítulo 5

• Las decisiones con el ordenador.
• Las órdenes de detención del programa.

ESQUEMA DE CONTENIDO

5.0 OBJETIVOS

El objetivo principal de este capítulo es aprender a realizar decisiones con el ordenador. Para ello, aprovecharemos los conocimientos adquiridos en el volumen anterior sobre comparaciones binarias, y los aplicaremos a la instrucción IF... THEN que estudiaremos aquí. La toma de decisión con ordenador es un punto muy importante, pues sin ella el ordenador no podría realizar tareas complejas.

Además, aprenderemos también a detener el programa en puntos determinados mediante las instrucciones END y STOP.

Por otra parte veremos el proceso que se sigue para elaborar un programa un poco largo. Estas técnicas de diseño serán ampliadas en el capítulo 7.

5.1 DECISIONES CON EL ORDENADOR

Hemos citado en el tomo anterior que el ordenador podía tomar decisiones, y hemos desarrollado el tema de cómo hay que formular preguntas al ordenador. Hasta el momento sólo hemos utilizado el ordenador como una máquina de calcular, vamos a intentar que haga algo más. La posibilidad de tomar decisiones es lo que distingue a un ordenador de una máquina de calcular.

El método de tomar decisiones

Antes que nada, analicemos un poco qué significa tomar decisiones. Consideremos varios ejemplos, en los que nos fijaremos principalmente en el método a seguir más que en la decisión a tomar:

1. Debemos elegir un sitio para ir de vacaciones.

La primera cosa que hacemos es listar las alternativas que tenemos, por ejemplo:

— Playa.

— Montaña.

— Alta montaña.

— Viajar.

—Quedarnos en casa y descansar.

La segunda cosa que hacemos es utilizar algún criterio para descartar o elegir alguna de estas alternativas. Por ejemplo, tomamos el criterio económico y elegimos la última alternativa, la de quedarnos en casa, porque es la que supone un menor gasto.

A veces utilizaremos una combinación de criterios; por ejemplo, además del criterio económico se utiliza el criterio de placer.

En el ejemplo, la combinación de economía y placer nos puede llevar a elegir la alternativa de viajar.

2. Debemos elegir un menú en un restaurante.

La lista de alternativas no la confeccionamos en este caso nosotros sino que nos la ofrece el camarero con una carta que contiene una larga lista de platos.

La confección de nuestro menú se hará según tres criterios básicos: el apetito, la imaginación del placer que producirá un plato determinado y el criterio económico.

Probablemente excluimos de nuestra comida aquellos platos que no nos apetecen, o bien los excluimos porque su precio es demasiado caro. En cambio hay otros platos que el criterio fundamental puede ser el gusto.

¿Cómo se toma una decisión?

En conclusión, elaborar una decisión consiste en establecer los pasos siguientes:

a) Una lista de alternativas. Las opciones de vacaciones o la lista de platos.

b) Analizarlas bajo unos criterios determinados. Utilizar la combinación de criterios económicos y de placer en el caso de la elección de vacaciones. O la combinación de criterios de apetito, gusto y económico en el caso del restaurante.

c) Valorar cada una de las alternativas con los criterios establecidos. De la lista de lugares donde vamos a pasar las vacaciones, repasamos cada opción y establecemos un valor, que es el que nos dirá qué hay que decidir. Un ejemplo de nuestras cavilaciones puede ser el siguiente:

La playa no me produce ningún placer especial, puedo ir cada día del año y no debe ser especialmente en vacaciones; además es caro. La montaña no resulta caro, pero no es demasiado atractivo: gozaré de una tranquilidad absoluta pero no me divertiré. La alta montaña, presenta inconvenientes económicos pues preciso de un equipo realmente caro que no tengo; por otra parte, el placer no será muy grande ya que estoy en baja forma y la alta montaña requiere un esfuerzo físico. Viajar puede ser una opción interesante, podría ir aquí o allí, sale un poco caro pero puede reducirse el número de días. Ciertamente, quedarse en casa es lo que más barato sale, pero no rompo el ritmo monótono que he llevado durante todo el año; podré dormir mucho, podré hacer aquel hobby que tengo tantas ganas de practicar, pero desde el punto de vista del placer no romperé el ritmo lo suficiente.

d) Elegir la alternativa cuyo resultado sea más favorable. Finalmente, después de las consideraciones anteriores, que en ciertas personas pueden durar mucho, decido elegir la alternativa del viaje, en el caso de las vacaciones y el menú que voy a tomar en el caso del restaurante.

En la vida corriente el proceso de valoración de una decisión nunca está demasiado claro, pero a la postre siempre elegimos una alternativa aunque sea por indiferencia o exclusión de las demás.

El proceso de toma de decisiones en un ordenador, significa exactamente lo mismo, pero con unas matizaciones que convendrá discutir.

En efecto, el proceso de toma de decisión en el ordenador se fundamenta en los pasos siguientes:

1. Hay que elaborar una lista de alternativas.

Este punto es muy importante pues en el ordenador, a diferencia del proceso mental, sólo se considerarán las alternativas que listemos, es decir, que determinemos antes del proceso de decisión.

Generalmente en el proceso mental rehacemos la lista varias veces y sobre la marcha.

2. Establecer unos criterios para valorar cada alternativa.

A diferencia del proceso mental, en donde los criterios suelen ser vagos e imprecisos, en el proceso con el ordenador estos criterios deben ser cuantitativos y exactos.

Por ejemplo, en la elección del menú, utilizamos un criterio de que el plato no sea demasiado caro; esta definición es demasiado imprecisa para el ordenador y debemos cambiarla a «se considera un plato caro el que sobrepasa, por ejemplo, 1000 unidades monetarias».

3. Realizar las acciones correspondientes a la decisión que se ha tomado.

La diferencia entre el proceso mental y el del ordenador es que cuando se emprenden las acciones correspondientes a unas decisiones no es posible volverse atrás y el ordenador las realizará indefectiblemente.

Aunque el estudio del proceso global de toma de decisiones es una cosa muy complicada que constituye una ciencia en sí misma, estos procesos siempre se subdividen en decisiones muy elementales, de manera que paso a paso se toman decisiones muy complicadas.

En esta Enciclopedia se estudiarán los procesos de decisión más elementales para introducirnos en la mecánica de operación del ordenador y sobre todo para realzar la diferencia más importante entre cálculo puramente y tratamiento de la información. Es muy difícil imaginar un programa útil sin que existan diversas alternativas de tratamiento.

Las figuras 1 y 2 del capítulo 4 han mostrado un proceso en donde la toma de decisiones elementales es importante para la determinación de los operadores relacionales entre variables textuales.

Los pasos 2, 3, 4, 5 y 6 de la figura 2 del capítulo 4 son ejemplos de decisión elemental que toma el ordenador.

La construcción

SI......ENTONCES......EN CASO CONTRARIO

es la manera elemental que el ordenador utiliza para tomar decisiones.

SI — Hace una pregunta binaria; es decir, una pregunta entre dos opciones.

ENTONCES — Es la orden que recibe el ordenador de tomar una decisión, si en la pregunta hecha en la pregunta binaria, es decir en el SI, tiene respuesta afirmativa.

EN CASO CONTRARIO — Es la orden que recibe el ordenador de tomar otra opción distinta si a la pregunta binaria del SI se da una respuesta negativa.

La instrucción de bifurcación

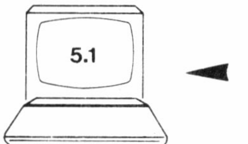
Este tipo de instrucción se denomina bifurcación, y es necesaria para cualquier lenguaje de programación y, por lo tanto, necesaria para cualquier UCP (Unidad Central de Proceso).

La última parte de la construcción, es decir, la parte EN CASO CONTRARIO es opcional y no todos los BASIC la incorporan.

Observe que este tipo de opciones es frecuente en la vida ordinaria. Por ejemplo, cuando decimos: SI mañana llueve ENTONCES me quedaré en casa, EN CASO CONTRARIO saldré al campo.

Con la construcción estudiada se permiten realizar unas instrucciones de cálculo, por ejemplo, sólo en unas circunstancias determinadas, o lo que es lo mismo, se evitan ciertas instrucciones innecesarias o incorrectas según las circunstancias del propio cálculo.

Pasemos al estudio de cómo se codifica este tipo de instrucción en lenguaje BASIC.



5.2 LA INSTRUCCION IF ... THEN ...

La instrucción de bifurcación en el lenguaje BASIC consiste en

IF (pregunta binaria) THEN (sentencia)

ésta es la imagen anglosajona de la instrucción que hemos visto:

SI (pregunta binaria) ENTONCES (sentencia)

en efecto, el inglés IF se traduce por SI y THEN se traduce por ENTONCES.

¿Cómo se escribe?

Veamos varios ejemplos de utilización en modo de ejecución inmediato

```
IF 3 = 5 THEN PRINT "Hola"
```

como 3 es distinto de 5, la acción de imprimir Hola no se ejecuta. Se debe identificar bien que la parte 3=5 es la pregunta binaria que condiciona la ejecución de la instrucción PRINT «Hola».

```
IF 3=3 THEN PRINT "Hola"
```

en este caso, la respuesta a la pregunta que 3 es igual a 3 es SI, por lo tanto, en la pantalla aparecerá la palabra Hola.

```
IF "Garcia" <> "Perez" THEN PRINT
"No tienen el mismo nombre"
```

La diferencia en este caso es que utilizamos variables textuales para la pregunta binaria. Como realmente son nombres diferentes el mensaje aparecerá en pantalla.

```
IF "Perez" < "Garcia" THEN PRINT
"Estan en orden lexicografico"
```

en este caso, no aparecerá el mensaje pues realmente Pérez es posterior a García, es decir mayor (en orden lexicográfico) que García.

```
IF 3 < 5 THEN  LET DIF= 5-3 : PRINT
"La diferencia es ":DIF
```

Detrás del THEN pueden colocarse diversas instrucciones

En este ejemplo se muestra como detrás del THEN efectivamente se puede colocar una sentencia, es decir, varias instrucciones una detrás de la otra. (El DIF que aparece detrás del LET es simplemente una variable de tres letras. Podía haber sido sólo una letra, por ejemplo D).

Concentremos, en primer lugar, nuestra atención en el modo de construcción de la instrucción.

1. La palabra IF no es un verbo, como en las instrucciones que hemos visto hasta ahora, por ejemplo, PRINT (Imprimir), LET (Sea), INPUT (Entre), GOTO (Vaya a). Sin embargo, incluye la acción a realizar detrás de THEN.

2. Incluye entre la palabra IF y el THEN una pregunta binaria basada en los propios datos del programa. Es decir, toma un dato, coloca un operador relacional y otro dato.

Concentremos ahora nuestra atención en el comportamiento de la instrucción.

La pregunta es la que controla la ejecución de la acción que va detrás del THEN. Sólo si la respuesta a la pregunta es SI, se realiza la acción que va detrás del THEN. En definitiva, esta pregunta decide si la acción detrás del THEN se realiza o no.

¿Cómo se computa el IF?

La decisión que toma esta instrucción es muy elemental, pero constituye la base de decisiones más complicadas, que se compondrán de combinaciones de decisiones elementales de este tipo.

Para clarificar este comportamiento tecleemos el programa siguiente:

```
NEW
10 INPUT A
20 IF A < 5 THEN PRINT "Hola"
30 GOTO 10
```

Al enviar el comando RUN, entremos diversos valores de A, por ejemplo,

Entremos 3   Se imprime Hola

Entremos 7	NO se imprime Hola
Entremos −7	Se imprime Hola
Entremos 0	Se imprime Hola
Entremos 100	NO se imprime Hola

Para acabar la ejecución del programa debemos teclear el BREAK o tecla de interrupción del programa. Probablemente el programa está aguardando en la instrucción 10 INPUT A, y no todas las máquinas interrumpen la ejecución en estas condiciones, entonces cometa un error, por ejemplo, teclee García y después de la tecla de fin de línea aparecerá un error pues las letras García no tienen significado numérico. Por la lección de Prácticas usted ya sabe cómo funciona en su máquina. Si no lo recuerda, repáselo. .

Finalización controlada de un programa

Esta finalización del programa no es elegante, la instrucción permite que la finalización de los programas esté programada en el sentido de control de finalización por parte del operador en una sentencia determinada.

Modifiquemos el programa anterior según el esquema siguiente:

```
NEW
10 INPUT A
20 IF A < 5 THEN PRINT "Hola"
30 IF A<>0 THEN GOTO 10
```

el programa esencialmente es el mismo que el anterior, sólo se ha sustituido la instrucción 30 por otra instrucción de bifurcación.

El análisis de la instrucción 30 muestra que el proceso es el mismo que en el programa anterior mientras la variable A sea distinta de 0, pues la sentencia 30 hará la instrucción detrás del THEN hasta que A tenga el valor 0. La instrucción detrás del THEN es GOTO 10, que es la misma que en el programa anterior.

Sin embargo cuando el valor de la variable A sea cero, aparte de imprimir Hola, pues es menor que 5, no volverá a la sentencia 10 ya que no ejecutará la instrucción GOTO 10 que está detrás del THEN de la sentencia 30 y el programa acabará normalmente pues no tiene más instrucciones a ejecutar. La pregunta A<>0 es NO cuando A vale 0; por lo tanto, no se ejecuta la parte de detrás del THEN.

En resumen la norma de comportamiento de la instrucción IF ... THEN ... es que si la respuesta a la pregunta binaria es NO, no se ejecuta la sentencia que está detrás del THEN.

Consideremos otro ejemplo, que pretende analizar lo que se puede poner detrás de la parte THEN de la instrucción.

```
NEW
10 LET N=0
20 INPUT A
30 IF A < 5 THEN LET N=N+1: PRINT
"HE DICHO Hola ";N;" veces"
40 IF A <> 0 THEN GOTO 20
```

El ejemplo es una variación del programa anterior, hay que destacar como diferencia el hecho que detrás del THEN de la sentencia 30, realiza una verdadera sentencia, es decir más de una instrucción.

Sigamos un poco la ejecución:

Entremos un 3. Imprimirá HE DICHO Hola 1 veces

Entremos un 7. No imprime nada

Entremos un −5. Imprime HE DICHO Hola 2 veces

Entremos un 100. No imprime nada

Siga entrando los números que desee.

Después de introducir el 0, el programa se parará.

Vemos, por lo tanto, que detrás del THEN pueden colocarse diversas instrucciones que se ejecutan una detrás de otra si la pregunta que precede al THEN tiene una respuesta de SI.

La parte THEN puede incluir otro IF

Incluso una sentencia que puede ir detrás del THEN es otra sentencia IF; por ejemplo, consideremos un programa que quiere contar los números positivos que entremos que están comprendidos en el intervalo de 50 a 100.

El programa funciona leyendo varios números uno detrás de otro. La condición para finalizarlo es entrar un número negativo. Por lo tanto, puede entrar los números que quiera que aparentemente la máquina no realizará ninguna acción sobre la pantalla, excepto cuando entre un número negativo que la máquina le imprimirá la cantidad de números que se han entrado en el intervalo de 50 a 100.

Al entrar el texto se dará cuenta de que las primeras instrucciones, concretamente desde la 10 a la 90 son comentarios que constituirán la documentación del programa. En las líneas 10 a 30 se define el objetivo del programa. Las sentencias 40, 50 y 60 indican el modo de entrar los datos. Finalmente la 70, 80 y 90 nos dan indicaciones de cómo se organizan las variables del programa.

Es muy aconsejable que en cualquier programa que realicen, incluyan mediante comentarios estos datos, ya que después de un tiempo nos es difícil recordar lo que hace el programa.

```
NEW
10 REM Se entran una serie de numeros que
20 REM se quieren clasificar en los que estan
30 REM el intervalo de 50 a 100
40 REM Los numeros a considerar se entran uno
50 REM detras de otro y se colocan sucesivamente
60 REM en la variable A.
70 REM El contador de numeros entrados es N
80 REM El contador de numeros que estan en el intervalo es NI
90 REM Un numero negativo termina el proceso de contaje.
100 LET N = 0 : LET NI = 0
110 INPUT A
120 IF A<0 THEN GOTO 160
130 LET N = N + 1
140 IF A>=50 THEN IF A<=100 THEN LET NI = NI + 1
150 GOTO 110
160 PRINT "Ha entrado ";NI;" numeros en el intervalo"
170 PRINT "Ha entrado ";N-NI;" numeros fuera del intervalo"
```

Figura 1: Seguimiento del programa de clasificación de números.

PASO	A	NI	N	Observaciones
100	?	0	0	
110	3	0	0	
120	3	0	0	Respuesta NO
130	3	0	1	
140	3	0	1	Es menor que 50
150	3	0	1	
110	65	0	1	
120	65	0	1	Respuesta NO
130	65	0	2	
140	65	1	2	Está en el intervalo
150	65	1	2	
110	67	1	2	
120	67	1	2	Respuesta NO
130	67	1	3	
140	67	2	3	Está en el intervalo
150	67	2	3	
110	78	2	3	
120	78	2	3	Respuesta NO
130	78	2	4	
140	78	3	4	Está en el intervalo
150	78	3	4	
110	23	3	4	
120	23	3	4	Respuesta NO
130	23	3	5	
140	23	3	5	Menor que 50
150	23	3	5	
110	154	3	5	
120	154	3	5	Respuesta NO
130	154	3	6	
140	154	3	6	Mayor que 100
150	154	3	6	
110	−1	3	6	
120	−1	3	6	Salta a 160
160	Imprime : Ha entrado 3 números en el intervalo			
170	Imprime : Ha entrado 3 números fuera del intervalo			

La selección de un intervalo mediante un IF detrás del THEN

Analicemos el comportamiento del programa.

La sentencia 100 inicializa el contador de números N (números entrados) a cero, y el contador de números que están en el intervalo NI, también a cero.

Las sentencias 110 hasta la 150 constituyen la entrada de datos. La instrucción 120 es la que termina esta entrada de datos cuando un número es negativo; es decir, cuando la pregunta A<0 es cierta, se realiza la parte THEN de la instrucción y el programa va a la línea 160, fuera ya de la parte dedicada a la entrada de datos.

La instrucción 130 incrementa el contador de números entrados. La instrucción 140 es la más interesante del programa, es la instrucción que decide si un número entrado está en el intervalo. Esta sentencia tiene la característica de tener otro IF en la parte THEN del IF. Esto ilustra una nueva posibilidad cuyo funcionamiento comentaremos con profundidad. La instrucción 150 acaba la parte de entrada de datos y nos recicla a la obtención de un nuevo dato, mediante la instrucción GOTO 110.

Las instrucciones 160 y 170 son la impresión de resultados, se imprime el contador NI y la diferencia N-NI que nos da el número de valores que no están en el intervalo.

Analicemos ahora el comportamiento de la línea 140.

Supongamos que A vale 60. Como la respuesta es SI a la primera pregunta A>= 50, se ejecuta la sentencia que está detrás del primer THEN. Nos encontramos ahora con la pregunta del segundo IF, como la respuesta es SI, A<=100 ya que 60 es menor que 100, se ejecuta la instrucción que está detrás del segundo THEN, que consiste en incrementar el contador NI. Este comportamiento ocurrirá igual para todos los números entre 50 y 100, ambos inclusive.

Supongamos que A vale 10, un número menor que 50, como la respuesta es NO a la pregunta del primer IF no se ejecutará la instrucción detrás del primer THEN y en consecuencia ni se planteará la pregunta del segundo IF.

Supongamos que A vale 150, un número mayor que 100, la respuesta a la pregunta del primer IF, A>=50 será afirmativa por lo que se ejecutará la instrucción detrás del THEN. Esta instrucción es otro IF, en el cual la pregunta es A<=100, como 150 es mayor que 100, no se ejecutará la parte de detrás del segundo THEN.

En consecuencia, todos los números que no están en el intervalo no incrementarán el contador NI.

Realicemos un seguimiento del programa en la muestra de números 3, 65, 67, 78, 23, 154 a través de la tabla de la figura 1. (Recuérdese que si entra estos números por el teclado, sólo imprimirá en pantalla cuando meta un número negativo.) Veamos ahora otra versión de un programa que resuelve el mismo problema.

La diferencia con la versión anterior está en que aquí se utilizarán los contadores NI, para números en el interior del intervalo, y el contador NO para números que no están en el intervalo.

En este ejemplo, prescindimos de la parte de la documentación para resaltar más el programa, ya que la explicación es muy completa en el texto del mismo.

La instrucción 10 sirve para inicializar los contadores NI y NO a cero.

Las instrucciones 20 a 90 constituyen el proceso de entrada de datos. La instrucción 30 acaba esta parte de entrada de datos enviando el control a la línea 100 cuando se entra un número negativo.

Las instrucciones 100 y 110 imprimen los resultados, en este caso, se escriben directamente los contadores NI y NO.

El núcleo de la clasificación son las instrucciones 40 a 90.

```
10 LET NI = 0 : LET NO = 0
20 INPUT A
30 IF A < 0 THEN GOTO 100
40 IF A < 50 THEN GOTO 80
50 IF A > 100 THEN GOTO 80
60 LET NI = NI + 1
70 GOTO 20
80 LET NO = NO + 1
90 GOTO 20
```

```
100 PRINT "Ha entrado ";NI;
    " numeros en el intervalo"
110 PRINT "Ha entrado ";NO;
    " numeros fuera del intervalo"
```

Otro modo de escoger un intervalo

La instrucción 40 envía los números menores de 50 a la instrucción 80 en donde se incrementa el contador de números que no están en el intervalo. La instrucción 50 envía a 80 aquellos números que sobrepasan el valor 100, para incrementar también el contador de elementos fuera del intervalo, es decir el contador NO.

Sólo si el número es mayor que 50 y menor que 100 llega a la instrucción 60, en donde se incrementa el valor del contador de números que están en el intervalo, es decir el contador NI.

Después de las instrucciones 20 y 80 se envía a leer un nuevo número.

En el momento de leer el número negativo se va a imprimir cada uno de los contadores. A diferencia de la versión anterior el contador de números de fuera del intervalo no se encuentra por diferencia.

Es interesante presentar otra versión distinta de las anteriores, para resolver el mismo problema. Esta situación es muy corriente en programación; un mismo problema se puede resolver de muchas maneras distintas; mientras todas las versiones son correctas, lo que importa es el resultado y no la forma de escribirlo.

```
10 LET NI = 0 : LET NO = 0
20 INPUT A : IF A<0 THEN GOTO 60
30 IF A< 50  THEN LET NO = NO +1 : GOTO 20
40 IF A>100  THEN LET NO = NO + 1 : GOTO 20
50 LET NI = NI + 1 : GOTO 20
60 PRINT "Ha entrado ";NI;
   " numeros en el intervalo"
70 PRINT "Ha entrado ";NO;
   " numeros fuera del intervalo"
```

La introducción de múltiples instrucciones en cada paso del programa, reduce el número de líneas totales pero esencialmente no modifica en nada el funcionamiento.

Observe que aunque existen dos instrucciones LET NO = NO + 1, sólo se pasa por una de ellas si el número está fuera del intervalo; en efecto, si el número es menor que 50 se incrementa NO en 30 y se va a leer un nuevo número mediante el GOTO 20, es decir ya no pasa por la instrucción 40.

La instrucción IF puede hacer que ciertas instrucciones no se ejecuten siempre

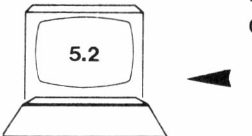
Observe que con la instrucción IF ... THEN ... es posible que algunas instrucciones del programa no se ejecuten durante la realización de un programa, por ejemplo en el programa anterior si todos los números que se entran son menores que 50, las instrucciones 40 y 50 no se ejecutan nunca.

En el momento de probar un programa hay que realizar las pruebas de tal manera que se pase alguna vez por todas las instrucciones con el fin de comprobar su buen funcionamiento.

Esto que parece muy sencillo no lo es tanto cuando el programa es largo y tiene muchas alternativas.

Para finalizar es necesario un comentario sobre la instrucción IF, dado que muchos manuales y libros se refieren a ella como salto condicionado en contraposición a la instrucción GOTO que es un salto incondicional.

La razón de este nombre proviene que el IF interrumpe el flujo normal de instrucciones según unas condiciones mientras el GOTO sí realiza propiamente un salto. Sin embargo, en BASIC este nombre no tiene demasiado sentido pues no necesariamente se produce un salto; después del THEN pueden existir más instrucciones que el simple GOTO, aunque probablementen, en la práctica la instrucción más común detrás del THEN sea precisamente un GOTO.



5.3 LA INSTRUCCION END Y STOP

La finalización de los programas en muchos dialectos del BASIC se realiza mediante unas instrucciones específicas de parada.

En este curso ya hemos estudiado la interrupción de un programa con la instrucción BREAK. Este método de parar un programa no debe ser el corriente y sólo debe utilizarse en el caso de preparación de un programa, cuando los errores no están bien depurados y es necesario interrumpirlo a la fuerza.

También se ha estudiado la instrucción STOP como método de parada en un punto específico de un programa. En este apartado volveremos a repasar para concretar sus diferencias con la instrucción END.

Finalización normal de un programa

La finalización normal de un programa se hace mediante la instrucción END, que significa FIN. La manera de escribir esta instrucción es muy simple, por ejemplo, veamos algunas formas.

```
10 END

10 IF A<0 THEN END

10 IF A<0 THEN PRINT "He acabado": END
```

Estos ejemplos no ilustran un programa; sólo pretenden mostrar la utilización de la instrucción en diversos puntos de las instrucciones que componen un programa.

Recordando el segundo ejemplo del apartado anterior

```
NEW
10 INPUT A
20 IF A < 5 THEN PRINT "Hola"
30 IF A <> 0 THEN GOTO 10
40 END
```

se ha añadido la instrucción 40 con un END.

De hecho esta instrucción puede considerarse superflua en este caso, pues el BASIC se acaba correctamente cuando se le acaban las instrucciones. Sin embargo, la colocación del END indica al lector del programa que no van más instrucciones y que es intención del programador acabarlo en aquel punto.

Tenga en cuenta que cuando Ud. haga un programa tendrá muy claro todo lo que el programa hace, pero con toda seguridad después de tres meses de haberlo escrito, esta claridad de ideas que posee en el momento de escribirlo habrá desaparecido y toda ayuda procedente del propio texto del programa tendrá un valor muy importante.

Por esta razón es aconsejable que cuando desarrolle programas utilice las instrucciones REM y las instrucciones END para colocar indicaciones claras en las instrucciones propiamente dichas.

La utilidad de la instrucción END es más clara en los programas que no acaban al final de todo, por ejemplo, consideremos un programa que cuente las letras de un texto, incluidos los blancos, y que queremos que se finalice cuando introduzcamos la palabra FIN.

```
NEW
10 INPUT A$ : IF A$ = "FIN" THEN END
20 PRINT "Longitud: ";LEN(A$)
30 GOTO 10
```

en la sentencia número 10 la última instrucción es END que nos indica cómo se acaba el programa.

En resumen, la instrucción para acabar un programa correctamente es la instrucción END, que puede aparecer en los más diversos lugares del programa.

El STOP como ayuda para probar un programa

La instrucción STOP se utiliza de manera idéntica a la instrucción END e incluso su comportamiento es el mismo.

Sin embargo, las dos instrucciones presentan una diferencia de comportamiento que es importante mencionar.

Cuando un programa se finaliza mediante una instrucción STOP es posible continuarlo mediante el comando CONTINUE, mientras en la instrucción END esto no es posible.

En este modo de comportamiento el STOP se parece mucho más a un BREAK que a un END, la diferencia con el BREAK es que se controla desde el programa mismo el momento de la parada.

Cuando se aprieta la tecla de interrupción de programa (BREAK), no sabemos dónde vamos a pararlo exactamente; en cambio, mediante la instrucción STOP sólo se parará el programa en el punto donde se encuentre la instrucción. Este punto suele ser clave para determinar el buen funcionamiento del programa.

Esta instrucción no debe usarse en programas que están correctos y acabados; sólo debe utilizarse durante el período de pruebas de un programa para efectuar detenciones programadas en puntos donde interesa observar el contenido de las variables, ya que después de esta investigación podemos arrancar de nuevo el programa en la instrucción siguiente al STOP que lo ha parado.

Consideremos el ejemplo siguiente, que consiste en entrar dos números A y B, e ir incrementando A hasta que A y B sean iguales.

```
NEW
10 INPUT A
20 IF A <= 0 THEN END
30 INPUT B
40 LET D = 0
50    LET A = A + 1
60    LET D = D + 1
70    IF D >= 1000 THEN STOP : LET D = 0
80    IF A<>B THEN GOTO 50
90 PRINT "Los numeros ";A;" y ";B;"
   ya son iguales "
1000 GOTO 10
```

La condición de finalización esperada por el programador es que cuando el primer número es cero o negativo el programa acabe correctamente. Esto se expresa en la instrucción 20.

Sin embargo debe darse cuenta, que cuando se introduce un número B más pequeño que A, ya sea por error o por desconocimiento, el programa no terminaría nunca ya que al ir incrementado A lo que hacemos es alejarnos cada vez más del valor de B. En términos del programa ejecutaríamos indefinidamente las sentencias 40 a 70.

Con el fin de prevenir esta probabilidad hemos introducido el contador D que cada 1000 veces que se incremente ejecutará una instrucción STOP, de tal manera que podemos investigar si el programa finalizará o no, pues comprobaremos si A es ciertamente menor que B; en caso de que lo sea, apretaremos la instrucción CONTINUE para que se produzca el cálculo correcto del programa.

Si A es mayor que B, volveremos a dar la instrucción RUN, pero cuidando de entrar los valores correctos de A y de B.

Después de estas explicaciones del funcionamiento hagamos los experimentos siguientes:

1. Una vez tecleado el programa lo ejecutaremos mediante una instrucción RUN.

Introduzcamos como valor de A el 7 y como valor de B el 670. Tecleemos a continuación un BREAK, el programa se interrumpirá no sabemos exactamente dónde pero lo podremos averiguar mediante las instrucciones de ejecución inmediata

PRINT A,B,D

A continuación tecleamos la instrucción CONTINUE y el programa se reanudará. Al cabo de un rato imprimirá el mensaje de que A ha alcanzado a B y nos pedirá otro valor de A.

Si introducimos el valor cero el programa finalizará correctamente.

2. Tecleemos otra vez el comando RUN

Introduzcamos el valor de A de 145 y a continuación el valor de B de 12. En este caso el programa no debería pararse nunca pues A es mayor que B y, al ir incrementándolo, lo único que conseguimos es alejarnos del valor de B.

Esperemos un rato, y observaremos que el programa se para, aparecerá un mensaje en pantalla de STOP. Esto nos indicará que después de haber realizado 1000 veces las instrucciones 40 a 70 del programa, el programa no ha terminado aún.

Mediante la instrucción inmediata

PRINT A,B,D

encontramos unos valores de 1145 para A, 12 para B y para D de 1000. Nos damos cuenta inmediatamente que A es mayor que B y que los datos son incorrectos.

3. Tecleemos otra vez el comando RUN

Introduzcamos ahora para A el valor de 100 y para el de B de 2345.

Deberemos esperar un rato y a continuación aparecerá en pantalla el mensaje de STOP.

Con la instrucción de modo inmediato

PRINT A,B,D

aparecerá el valor 1100 para A, el valor de 2345 para B y para D el valor de 1000.

El valor de A es menor que B, por lo tanto, teclearemos el comando CONTINUE y el programa proseguirá correctamente.

Al cabo de otro rato, se producirá otra parada y aparecerá en pantalla el STOP, que indicará que han pasado 1000 vueltas más sin que el programa se haya terminado.

Con la instrucción inmediata

PRINT A,B.D

observaremos que A tiene un valor de 2100, el de B de 2345 y el de D otra vez 1000, recuerde que después de la instrucción STOP, colocamos D a un valor de cero.

Volvemos a apretar la tecla de CONTINUE y el programa continuará correctamente ya que el valor de A es menor que el valor de B.

Finalmente el programa lanzará el mensaje de que A ha alcanzado el valor de B y nos pedirá un nuevo valor de A, que introduciremos como O y el programa finalizará correctamente.

Se puede impedir todo este mecanismo mediante una instrucción de decisión según se indica en esta nueva versión del programa, que consiste en evitar que se inicie el cálculo cuando el número B es menor que A, imprimiendo un mensaje que diga que B es menor que A.

```
NEW
10 INPUT A
20 IF A <= 0 THEN END
30 INPUT B
40 IF B<= A THEN PRINT "B es menor que A":
   GOTO 10
50   LET A = A + 1
60   IF A<>B THEN GOTO 50
70 PRINT "Los numeros ";A;" y ";B;
   " ya son iguales "
80 GOTO 10
```

La instrucción 40 impide que se empiece el incremento de A sin asegurarse de que B es mayor que A.

Esto no es una situación anormal. Casi siempre, la modificación de las condiciones, es decir, el depurado de los programas evitan las instrucciones STOP, que en definitiva sólo sirven para poder investigar el estado de la ejecución del programa.

El ejemplo que se ha estudiado es un poco forzado, pero debe reconocer que es precisamente la utilidad principal de la instrucción STOP la que impide encontrar un ejemplo en que las cosas sean incorrectas sin ponerlo de manifiesto en la explicación. Cuando Ud. desarrolle algún programa sin ninguna guía se encontrará con condiciones en las que no había pensado y es cuando debe utilizarse esta instrucción STOP para verificar todas las suposiciones que había hecho al diseñarlo.

La instrucción STOP seguida de un número de sentencia

Finalmente muchos dialectos de BASIC permiten que la instrucción STOP esté seguida de una expresión aritmética o textual que aparece en pantalla cuando esta instrucción se ejecuta.

La aplicación más importante de esta expresión es indicarnos dónde se ha parado el programa. Cuando se depura un programa se colocan diversas instrucciones STOP en los puntos claves del mismo. La expresión detrás del STOP permite distinguir en qué instrucción se ha parado entre las diversas instrucciones STOP que están esparcidas en el programa.

RESUMEN

El ordenador se distingue de la máquina de calcular porque es capaz de tomar decisiones aparte de calcular.

Tomar una decisión significa realizar los pasos siguientes:

1. Listar las alternativas posibles a la resolución de un problema
2. Establecer unos criterios de evaluación generales
3. Valorar cada una de las alternativas con los criterios establecidos en el paso 2

4. Escoger la alternativa que nos es más favorable

Cuando este proceso se realiza con el ordenador, es necesario conocer con precisión todas las alternativas antes de iniciar el proceso. Los criterios para evaluar cada una de las alternativas deben ser rigurosos y precisos; diremos que deben expresarse de forma matemática. Una vez se ha escogido el camino a seguir no es posible cambiar la decisión.

Todo proceso de decisión complejo se realiza mediante la ejecución de múltiples procesos elementales.

El proceso de decisión elemental consiste en formular las valoraciones mediante la construcción SI.....ENTONCES.....EN CASO CONTRARIO.....

Detrás del SI aparece una pregunta binaria, detrás del ENTONCES se dan las órdenes cuando la respuesta a la pregunta es afirmativa y detrás de EN CASO CONTRARIO se dan las órdenes cuando la respuesta a la pregunta es negativa.

Este tipo de instrucción existe en todos los ordenadores y se denomina instrucción de bifurcación.

En el lenguaje BASIC la instrucción que realiza la bifurcación es la que se codifica como

IF (pregunta binaria) THEN (sentencia)

La pregunta binaria más simple en BASIC es la que utiliza un operador relacional como los que se han estudiado en la lección.

Se debe recordar que detrás del THEN se puede colocar una sentencia, es decir, varias instrucciones separadas por los dos puntos. Esta sentencia a su vez puede contener otra instrucción IF.

En el aspecto sintáctico esta instrucción no empieza por un verbo (es la manera de dar órdenes), sino por una pregunta binaria para realizar la acción solamente cuando la pregunta binaria es afirmativa.

La introducción de las instrucciones IF...THEN en los programas hace que no siempre se efectúen todas las sentencias. Esto dificulta, las pruebas, pues hay que diseñarlas para que cada instrucción se ejecute una vez por lo menos.

La instrucción IF...THEN se suele denominar salto condicionado y la instrucción GOTO salto incondicional, pues se produce el salto sin ninguna condición.

La instrucción de finalización de un programa es la instrucción END. Esta instrucción es realmente útil cuando un programa no finaliza al final de las líneas escritas y se quiere indicar claramente que el programa finaliza en aquel punto.

Las otras instrucciones de parada (la tecla de interrupción BREAK y la instrucción STOP) son instrucciones que sólo deben utilizarse para depurar los programas, a diferencia de la instrucción END que significa una terminación correcta de los programas.

Según qué dialectos del BASIC, la instrucción STOP puede ir seguida de un número para indicar en qué lugar del programa se ha introducido la parada.

EJERCICIOS DE AUTOCOMPROBACION

Completar las frases siguientes:

1. El ordenador se distingue de una máquina de calcular porque toma ....

2. El primer paso para tomar una decisión es .... las alternativas.

3. En el ordenador la evaluación de los criterios de la decisión debe ser muy ....

4. Todo proceso de decisión .... se descompone en procesos de decisión elementales.

5. Se entiende por proceso elemental de decisión aquel que se expresa mediante la construcción SI .... ENTONCES una acción EN CASO CONTRARIO otra acción.

6. La instrucción de los ordenadores que permite el proceso de decisión elemental se denomina instrucción de ....

7. La instrucción END sirve para señalar la .... de un programa.

8. La instrucción STOP es una parada que permite .... el programa en el lugar donde se detiene.

9. La tecla de interrupción y la instrucción STOP sólo deben utilizarse para .... un programa.

10. En algunos dialectos del BASIC la instrucción puede llevar detrás un número o texto para informar de donde se .... el programa.

En las instrucciones siguientes rodee con un círculo la V si la instrucción es correcta o con una F si la instrucción es incorrecta.

11. IF LET A = 5 THEN GO TO 100 V F

12. IF A< 3 THEN STOP V F

13. IF A =A$+«5» THEN 5-4 V F

14. IF A$ < «3» THEN LET A$= A$ – 1 V F

15. IF A$ = STR$(B) THEN A$ = «Incorrecto» V F

Diga cuál es el valor de las variables que intervienen cuando se ejecutan cada una de las instrucciones siguientes:

16. LET A = 5 : IF A+3 < A–3 THEN LET A = 8

A =

17. LET A = –5 : IF A+3 < A–3 THEN LET A = 8

A =

18. LET A = 1 : LET B = 2 : IF B<2 THEN IF A=1 THEN LET A = B–A

A = B =

19. LET A = 2.45 : IF A> 2.5 THEN LET A = A + 1

A =

20. LET A$ = «5» : LET B$ = «A» : IF A$=B$ THEN LET A$=B$+« «+A$

A$= B$=

21. LET A$ = «m» : IF A$ <= «Z» THEN LET A$ = CHR$ (ASC(A$)+32)

A$=

22. LET A$ = «M» : IF A$ <=«Z» THEN LET A$ = CHR$(ASC(A$)+32)

A$=

23. LET A$ = «JUAN» : IF A$ ==« « THEN LET A$ = «INCO-RRECTO»

A$=

24. LET A=13 : IF A < 2*A THEN LET A= 3*A.

A =

25. LET A=−13 : IF A < 2*A THEN LET A = 3*A

A =

5.4 LA INSTRUCCION IF CON ELSE

En el ejemplo del orden lexicográfico del tomo anterior hemos utilizado la construcción

SI ...

ENTONCES ...

EN CASO CONTRARIO ...

En algunos dialectos del BASIC existe la instrucción que tiene la forma

IF (pregunta binaria) THEN (instrucción) ELSE (instrucción)

Los términos IF, THEN y ELSE son las palabras anglosajonas que se traducen como SI, ENTONCES y EN CASO CONTRARIO.

El comportamiento de esta instrucción es semejante a la del IF aunque presenta unas diferencias que vamos a comentar a partir del ejemplo siguiente. (Si su ordenador no tiene la instrucción ELSE no intente introducirlo; ya veremos cómo lo puede hacer en su caso).

```
10 INPUT A
20 IF A<0 THEN LET T$="NEGATIVO" ELSE LET
   T$="POSITIVO"
30 PRINT "EL NUMERO QUE HA ENTRADO ES ";T$
40 GOTO 10
```

La misión de la instrucción 20 es calcular si el número es positivo o negativo; la variable textual T$ se coloca a NEGATIVO si la respuesta a la pregunta binaria A<0 es SI, mientras que se coloca al valor POSITIVO si la respuesta es NO.

El comportamiento de la instrucción es realizar la instrucción que está detrás del THEN si la respuesta es cierta, y la de detrás de ELSE si la respuesta es falsa. **Tenga en cuenta que en el caso de que sea cierta la pregunta binaria nunca se realiza la instrucción que está detrás del ELSE de la misma manera que cuando la pregunta es falsa no se realiza la instrucción que está detrás del THEN.**

Es la instrucción COMPLETA de bifurcación

Este tipo de instrucción es una extensión natural de IF... THEN... y corresponde a la idea de la bifurcación completa; es decir, realizar instrucciones distintas en caso de respuesta afirmativa o negativa de la pregunta binaria.

Esta instrucción es un caso particular que parecería no es necesario tratar al estudiar el BASIC estandard. Sin embargo, la importancia de la construcción lógica SI.....ENTONCES.....EN CASO CONTRARIO es tan grande en el diseño de programas y en la toma de decisión que es obligado considerar esta construcción aunque el BASIC con que trabajamos no disponga de ella.

La solución consiste en tomar diversas sentencias para simular el comportamiento al tener que solucionar un programa sin la ayuda del ELSE. Para ello tomaremos el mismo ejemplo visto anteriormente, en el que, al introducir un número en el ordenador, si el número es menor que cero, debe escribir NEGATIVO y en caso contrario debe escribir POSITIVO.

Veamos cómo se escribiría este programa sin la ayuda del ELSE.

```
10 INPUT A
20 IF A<0 THEN GO TO 50
30 PRINT "El numero que ha entrado es positivo"
40 GO TO 60
50 PRINT "El numero que ha entrado es negativo"
60 GO TO 10
```

Ahora vea la figura 2 que contiene el mismo programa pero en él hemos recuadrado tres bloques:

— El primero, que aparece identificado por IF, plantea la pregunta binaria.

— El segundo, que aparece identificado por ELSE resuelve la parte correspondiente al ELSE, o lo que es lo mismo, lo que hace el programa cuando la respuesta a la pregunta IF es NO.

— El tercero, que aparece identificado por THEN, resuelve la parte del THEN; es decir, lo que hace el programa cuando la respuesta al IF es SI.

Vea ahora cómo funciona el programa. Cuando la pregunta binaria de la línea 20 es SI, porque el número que se ha entrado es negativo, el pro-

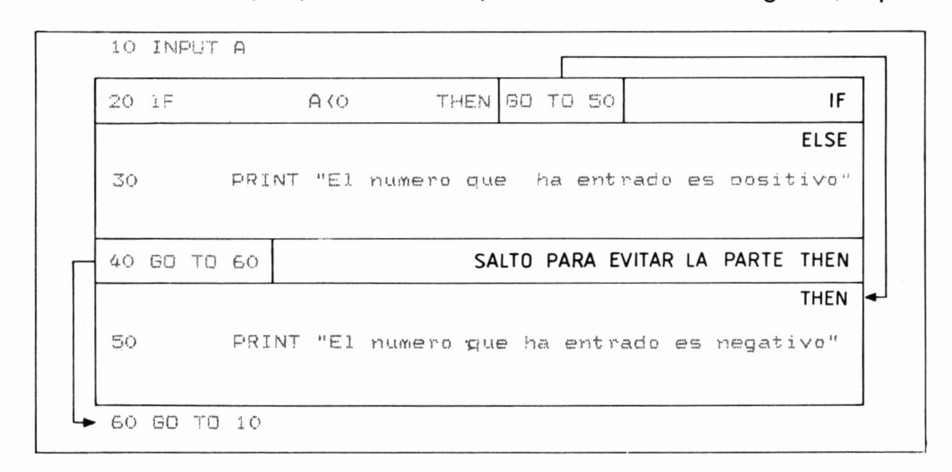


Figura 2: Construcción IF ... THEN ... ELSE, sin la utilización del ELSE.

grama nos remite a la línea 50 mediante el GO TO y, por tanto, el ordenador imprimirá en pantalla: «El número que ha entrado es negativo».

Si la respuesta a la pregunta binaria es NO, porque se ha entrado un número positivo, la parte que sigue al THEN del IF de la pregunta no se ejecuta y el ordenador sigue en la línea 30 e imprime en pantalla: «El número que ha entrado es positivo».

Esta parte correspondería al ELSE de la construcción general; precisamente la línea 40, mediante el GO TO envía la ejecución del programa a la línea 60 que es la finalización de la construcción. De este modo se impide que una vez entrado en la parte ELSE de la construcción no se ejecute la parte THEN ya que no tiene sentido ejecutar las dos partes, o se ejecuta una o la otra, pero nunca las dos.

El programa que hemos puesto es muy simple. En lugar de tener una sola instrucción después del THEN y del ELSE podría haber tenido un gran número de sentencias. Pero en todos los casos la estructura hubiera sido la misma.

La estructura IF...THEN...ELSE en la construcción de programas

Veamos otro ejemplo de programa con la construcción IF...THEN... ELSE. Consiste en llevar la cuenta de los cargos y abonos que se efectúan a un cliente y mantener el saldo después de cada operación.

Cuando hacemos una factura a un cliente, decimos que hacemos un cargo en su cuenta. Si por cualquier error se debe devolver dinero al cliente se dice que le hacemos un abono.

Esto se puede interpretar como llevar sumas separadas de números positivos y negativos. Los números positivos corresponden a los cargos, mientras que los números negativos corresponden a los abonos. El saldo es la diferencia entre los cargos menos los abonos, entre los números positivos y los números negativos.

El listado del programa se ofrece a continuación:

```
 10 LET TC=0:LET TA=0 : LET SA=0
100 INPUT "Cargo, Abono o Fin (c/a/f):",A$
110 IF A$="c" THEN LET O=1: GOTO 200
120 IF A$="C" THEN LET O=1: GOTO 200
130 IF A$="a" THEN LET O=2: GOTO 200
140 IF A$="A" THEN LET O=2: GOTO 200
150 IF A$="f" THEN GOTO 900
160 IF A$="F" THEN GOTO 900
170 GOTO 100
200 REM Entrada valida y distinta de fin
210 INPUT "Cantidad:",C
220 IF O<>1 THEN GOTO 300
230        REM Parte ELSE
240        LET TC = TC +C
250        LET SA = SA +C
260        GOTO 400
300        REM Parte THEN
310        LET TA = TA + C
320        LET SA = SA - C
400 REM Finalizacion de la construccion.
```

```
410 PRINT "Cargos";TAB(12);"Abonos";TAB(24);
    "Saldo"
420 PRINT TC;TAB(12);TA;TAB(24);SA
450 GOTO 100
900 REM Finalizacion del programa
910 END
```

La línea 10 del programa coloca todos los acumuladores de cargos (TD), abonos (TA) y saldos (SA) a cero.

De la línea 100 a la 170 corresponde el bloque de validación de la entrada. La línea 100 pide la entrada de la variable textual A$. Las líneas 110, 120, 130 y 140 comprueban si corresponde a la letra c o a, tanto en mayúsculas como en minúsculas. Si alguna de las preguntas es cierta se coloca la variable numérica 0 a 1 o 2 si corresponde a cargo o abono respectivamente y se envía la ejecución a la línea 200.

Las líneas 150 a 160 preguntan si la variable A$ contiene una f, tanto en mayúscula o minúscula. Si la respuesta es afirmativa se envía el control del programa a la línea 900 para la finalización del mismo.

Cualquier otra entrada se considera errónea y la línea 170 envía el programa a realizar la pregunta de nuevo.

De la línea 200 hasta la 400 está el bloque correspondiente a la construcción IF....THEN....ELSE que se quiere ilustrar.

En primer lugar se pregunta qué cantidad corresponde al cargo o abono, en cualquier caso se entra la cantidad como número positivo.

La línea 210 inicia la construcción con la pregunta binaria de si la variable 0 es distinta de 1. En caso afirmativo se envía el programa a la sentencia 300 que inicia la parte de THEN de la construcción. Si la respuesta es negativa se inicia la ejecución de las líneas 220 a 250.

En el bloque ELSE se actualiza el total de cargos y se les añade en forma positiva al saldo.

Se finaliza el bloque ELSE con la línea 260 que envía el control a la línea 400 para evitar la ejecución de la parte THEN.

En la parte THEN, de las líneas 300 hasta la 320 se calcula la parte THEN acumulando la cantidad entrada en el acumulador de abonos y disminuyendo el saldo en la misma cantidad.

El bloque que comprende la línea 400 hasta la línea 450 consiste en la escritura de los resultados después de la última operación.

La línea 900 y 910 finalizan la ejecución del programa.

La instrucción IF...THEN...ELSE con varias instrucciones

Finalmente hay que señalar que la mayoría de dialectos que permiten la construcción IF ... THEN .... ELSE sólo admiten una instrucción detrás del THEN y una instrucción detrás del ELSE, como en el caso de utilizar la construcción simple IF...THEN.

Si nos referimos al bloque de las líneas 220 a la 400 en algunos dialectos modernos del BASIC es posible escribirlo como

```
220 IF A<0 THEN LET TC=TC+C:LET SA=SA+C: ELSE
    LET TA=TA+C: LET SA = SA - C
```

Sin embargo, por el momento son más las excepciones que la regla. Para saber exactamente el comportamiento de estas instrucciones debe consultar el manual del usuario del tipo de dialecto de BASIC de que se trate.

Ahora ejecute el programa. Recuerde que «c» o «C» es para entrar los cargos, «a» o «A» para entrar los abonos y «f» o «F» para salir del programa. Tanto en la columna de cargos como de abonos le irá dando el acumulado. No se preocupe tampoco por el diseño en que le van saliendo los resultados.

5.5 EL CASO DE MUCHAS ALTERNATIVAS

Ya hemos visto en el tomo anterior que cuando queremos hacer una pregunta compleja como ¿De qué color es el coche?, debemos recurrir a la descomposición de la pregunta en múltiples preguntas binarias, es decir, debemos preguntar sucesivamente si el coche es blanco, es azul, es rojo, es verde, etcétera.

Debido a que este esquema aparece frecuentemente en programación, debemos ser sistemáticos en plantearlas y escribirlas en BASIC.

Veamos el ejemplo siguiente, que pretende realizar una operación aritmética entre dos números que hemos entrado.

En primer lugar se entran dos números y luego el programa pregunta por la operación aritmética que ralizará según la contestación que demos.

```
NEW
10  INPUT "El primer numero:",A
20  INPUT "El segundo numero:", B
30  INPUT "Indique la operacion a realizar (+,-,*,/):",A$
40  IF A$ <> "+" THEN GOTO 70
50      PRINT A+B
60      GOTO 180
70  IF A$ <> "-" THEN GOTO 100
80      PRINT A-B
90      GOTO 180
100 IF A$ <> "*" THEN GOTO 130
110     PRINT A*B
120     GOTO 180
130 IF A$ <> "/" THEN GOTO 160
140     PRINT A/B
150     GOTO 180
160 PRINT "La operacion que me pide es incorrecta"
170 GOTO 30
180 REM---- Fin de la decision para las alternativas
190 REM---- Calculemos otra operacion
200 GOTO 10
```

Se agrupan las sentencias que corresponden a un caso detrás de la pregunta

La estructura consiste en preguntar en los IF de las instrucciones 40, 70, 100 y 130 si no corresponde a una operación, dese cuenta que preguntamos si la operación NO es la que deseamos. Si la pregunta es afirmativa, es decir, no es la que deseamos, hacemos saltar el programa a la pregunta siguiente. Si la pregunta es negativa quiere decir que hemos en-

contrado la operación que buscamos, procedemos a realizarla y enviamos el programa al final de todas las preguntas, es decir en la instrucción 180.

Un comentario aparte merece la instrucción 160, que recoge el caso de que lo que hemos entrado, A$, no corresponde a ninguna operación. Es decir, que llegamos a ella cuando no metemos ningún operador de los previstos. Escribimos entonces un mensaje de error que programamos nosotros mismos y volvemos a leer una nueva operación.

Finalmente el programa nos vuelve a pedir dos nuevos números y una nueva operación.

Debemos advertir que podríamos haber realizado el programa con sentencias como

```
NEW
10  INPUT "El primer numero:",A
20  INPUT "El segundo numero:", B
30  INPUT "Indique la operacion a realizar (+,-,*,/):",A$
40  IF A$ = "+" THEN PRINT A+B: GOTO 90
50  IF A$ = "-" THEN PRINT A-B: GOTO 90
60  IF A$ = "*" THEN PRINT A*B: GOTO 90
70  IF A$ = "/" THEN PRINT A/B: GOTO 90
80  PRINT "La operacion que me pide es incorrecta": GOTO 30
90  REM---- Fin de la decision para las alternativas
100 REM---- Calculemos otra operacion
110 GOTO 10
```

que da como resultado un programa más corto y más compacto. Sin embargo, la forma como se ha presentado inicialmente está bien adaptada cuando las instrucciones a realizar para cada alternativa son muchas, que por otra parte suele ser el caso más corriente.

La norma general para la programación de diversas alternativas consiste en realizar las preguntas en forma negativa para saltarnos, en caso de que sea afirmativa la pregunta negativa, las instrucciones correspondientes a la alternativa que está en la pregunta.



5.6 FASES DE REALIZACION DE UN PROGRAMA

Diseño de un programa

En el capítulo 4 hemos estudiado cómo se podían escribir las instrucciones para determinar el resultado de la comparación de dos variables textuales. La instrucción se había redactado con un formato adecuado para la programación, pues las instrucciones se dividían en pasos y se utilizaban descripciones gramaticales con una precisión más elevada que el lenguaje corriente. La utilización de ciertas palabras en mayúsculas hacían énfasis sobre las entidades, números y letras, que tenían un significado importante en las reglas. Esta fase de escribir el mecanismo de solución del problema de forma más rigurosa que la que utilizamos en el lenguaje corriente, se denomina fase de *diseño* o de *análisis,* pues descomponemos de una manera más precisa y rigurosa los pasos a seguir.

Definición del problema

En la primera fase sólo hemos definido el problema de una manera vaga. A esta fase se le denomina *definición* del problema. En nuestro ejemplo, la definición del problema consiste en establecer que queremos com-

parar dos textos, y que la regla de obtención del resultado se ha expresado en la regla mencionada en la lección cuarta.

Codificación

La fase siguiente consiste en codificar este análisis o diseño en un lenguaje de programación. Esta fase se denomina *codificación,* y aunque es una fase importante, la calidad de un programa se determina en la fase de diseño. Si esta fase se efectúa correctamente, el análisis y el diseño sirven para realizar el programa en los más diversos tipos de lenguajes y máquinas.

Las distintas técnicas de diseño de programas serán objeto de un estudio más profundo en una lección posterior.

Ejemplo del orden lexicográfico

Volvamos ahora a ver la fase de codificación, y empecemos por la parte de definición de los datos que es necesario utilizar. Estos datos están listados en la figura 4 del capítulo 4. En ella se distinguen dos partes: los datos del problema, TEXTO1, operador y TEXTO2 y el resultado del problema. Estos datos nos indican una de las necesidades de nuestro programa que es la entrada de datos. El primer bloque del programa contendrá las construcciones necesarias para leer los datos.

Después de estas reflexiones ya podremos pensar en el tipo de instrucciones que vamos a utilizar. Antes de consultar el texto intente escribir en un papel aparte las instrucciones adecuadas.

```
10 REM FASE DE ENTRADA
20 INPUT "Primer texto";A$ : REM Entr. texto 1
30 INPUT "Segundo texto:";B$ : REM Entr. texto 2
40 PRINT "Entre uno de los operadores siguientes:"
50 PRINT "<, <=, <>, =, >, >="
60 INPUT O$
```

Las variables A$, B$ y O$ son los equivalentes en BASIC de las entidades TEXTO1, TEXTO2 y operador.

Probemos esta parte del programa. Para ello pulse la tecla RUN y vea que le va pidiendo los textos y el operador. Escriba para el primer texto GARCIA y para el segundo texto GARZA. Como operador elija por ejemplo >.

Para comprobar que el programa funciona dele la instrucción directa

PRINT A$,B$,O$

En la pantalla aparecerá lo que Ud. ha escrito.

Sigamos ahora desarrollando el programa.

El único problema que puede aparecer en este proceso de entrada de datos es que el usuario entre un operador que no corresponda con los datos que esperamos, en otras palabras, que no forme parte de la lista <, <=, <>, >, >=.

Filtrado de datos: Evitar que un dato llegue a la parte de cálculo en un contenido erróneo

Hay que evitar siempre que las variables que cuyo valor debe ser un elemento de una lista puedan llegar a otra parte del programa con un valor distinto de los admitidos, pues toda la lógica que hemos aplicado para la construcción de nuestro programa no se cumplirá, con el consiguiente funcionamiento equivocado del mismo.

En nuestro caso, es conveniente dotar al programa de un mecanismo para que después de salir de esta parte estemos seguros de que los valores de O$ es uno de los de la lista. Piense una solución antes de seguir con el estudio de la solución que le proponemos.

Una posible solución es (las instrucciones de la 10 a la 70 son las mismas que ya tenemos).

```
 70 IF O$="<" THEN GOTO 200
 80 IF O$="<=" THEN GOTO 200
 90 IF O$=">" THEN GOTO 200
100 IF O$="<>" THEN GOTO 200
110 IF O$="=" THEN GOTO 200
120 IF O$=">=" THEN GOTO 200
130 PRINT "Operador erroneo"
140 PRINT "Intentelo otra vez"
150 GOTO 60
```

La línea de razonamiento es comprobar si el símbolo que contiene O$ coincide con alguno de los símbolos esperados. Si se da esta coincidencia, se envía el programa a la instrucción 200 para que el programa continúe. Si no se trata de ninguno de los símbolos esperados se alcanza la línea 130, en donde el ordenador nos imprime el mensaje indicándonos que hay un error y nos da la oportunidad de corregir la entrada. Este proceso se repetirá hasta que se entre el símbolo correcto.

Fase de pruebas

Es interesante que probemos el funcionamiento de esta parte de programa por sí misma, sin acabar de escribir el resto. Este procedimiento se denomina fase de *pruebas* y *correcciones*. Para realizar un programa correctamente, es necesario emplear tiempo en probarlo para asegurar que no se han cometido errores en ninguna de las fases anteriores.

Sin embargo, hay que tener en cuenta que estas pruebas sólo nos demuestran que el programa es incorrecto pero nunca que es correcto. Esto es así porque no podemos asegurar que hemos probado todos los casos posibles. La verificación de la corrección absoluta de un programa es una tarea difícil y se parece mucho a las demostraciones matemáticas y no pretendemos introducirnos en este terreno, pues antes hay que tener una gran experiencia en programación.

Es muy conveniente por esta razón, probar sólo trozos pequeños de programa, a fin de asegurar con bastante certeza que el programa no tiene errores.

Si Ud. escribió esta parte del programa antes de estudiar el texto, probablemente hizo una versión de esta entrada de datos que no será la misma que la que hemos propuesto. Realice ahora un conjunto de pruebas como las siguientes, después de pulsar RUN.

— Entre como operador el signo más (+): debe dar error.

— Entre como operador el conjunto de letras ABC (cualquier texto se puede entrar en la instrucción 60): Debe dar error.

— Entre el operador igual (=): El programa debe finalizar. Escribamos RUN de nuevo y sigamos.

— Entre el operador menor o igual que (<=): El programa finaliza. Escribamos RUN.

— Entremos el operador mayor que (>): El programa finaliza. Escribamos RUN.

— Entremos el operador distinto de (<>): El programa finaliza. Escribamos RUN.

— Entremos el operador menor que (<): El programa finaliza. Escribamos RUN.

— Entremos el operador mayor o igual que (>=): El programa debe finalizar.

Si un programa supera este conjunto de pruebas quiere decir que es con gran probabilidad correcto.

Hasta ahora hemos desarrollado el tema de la entrada de datos, en nuestras instrucciones de referencia (Figuras 1 y 5 del capítulo anterior) que corresponde al punto 0 o inicio del programa. Pasemos a desarrollar la parte de programa más importante que es la que efectúa realmente la comparación.

Para proseguir, primero debemos determinar el nombre de las variables que nos servirán para fijar los datos intermedios:

Número de orden	N
NUM1	N1
NUM2	N2

Una vez establecida esta correspondencia, pasemos a traducir cada uno de los pasos. Para ello tenga delante la figura 1 del capítulo 4.

Paso 1:

```
200 LET N=1
```

Observe que hemos elegido la instrucción 200 en lugar de la 160. Lo hacemos porque no sabemos si hará falta añadir alguna instrucción intermedia. Por otra parte, es conveniente que cuando escriba programas, los distintos bloques queden bien diferenciados. Esto lo conseguiremos mediante el uso de instrucciones REM y saltándonos líneas en la numeración, lo cual nos permitirá señalar claramente un salto entre bloques.

Paso 2:

```
210 IF LEN(A$))=N THEN GOTO 240
220     LET N1=0
230     GOTO 250
240     LET N1=ASC(MID$(A$,N,1))
250 REM Fin paso 2
```

Observe que en este paso 2 utilizamos la función LEN para determinar la longitud del primer texto y la función ASC para saber el número de orden de la letra n-ésima del texto según el abecedario ASCII. La función MID$ selecciona la letra que se encuentra en la posición N dentro del texto. Esta operación nunca se realiza si N no es una letra del texto puesto que la instrucción 210 asegura que sólo se llega a la línea 240 cuando se cumplan las condiciones apropiadas. Es decir, cuando la respuesta al IF es afirmativa y consiguientemente se ejecuta la parte THEN.

La codificación que se ha mostrado es una traducción directa de lo que se ha escrito en el paso 2, sin embargo, las instrucciones siguientes producen el mismo resultado:

Paso 2 (Mejorado):

```
210 LET N1=0
220 IF LEN(A$))=N THEN LET N1=ASC
    (MID$(A$,N,1))
250 REM Fin del paso 2
```

En este caso, suponemos en principio que ya nos hemos pasado, colocamos un cero en N1 mediante la instrucción 210.

La instrucción 220 recalcula N1 sólo en el caso de que tenga sentido hablar de la letra de orden N.

Observe como el paso 2 ha quedado simplificado. Borre, por tanto, las líneas 230 y 240 escribiendo el número de línea y después el ENTER, o tecla de fin de línea.

El paso 3 es igual al paso 2, pero con el segundo texto y con N2; queda por lo tanto:

Paso 3:

```
260 LET N2 = 0
270 IF LEN(B$))=N THEN LET N2 = ASC
    (MID$(B$,N,1))
```

La traducción de los pasos 4, 5, 6, 7 y 8 es inmediata y es la siguiente:

```
Paso 4:   280        IF N1=0 THEN GOTO 500
Paso 5:   290        IF N2=0 THEN GOTO 500
Paso 6:   300        IF N1<>N2 THEN GOTO 500
Paso 7:   310        LET N = N+1
Paso 8:   320        GOTO 210
```

Observe que cuando traduce el paso 4, no sabe con exactitud dónde acabará el paso 8, por lo tanto, si utiliza un número de instrucción elevado, no tendrá problemas posteriormente en que esté mal calculado, y habrá creado una clara separación de bloques.

El bloque de cálculo

Antes de pasar a la traducción del paso 9, podemos repasar el bloque correspondiente a los pasos 1 a 8.

```
200 LET N = 1
210 LET N1=0
220 IF LEN(A$))=N THEN LET N1 = ASC
    (MID$(A$,N,1))
250 REM Fin del paso 2
260 LET N2 = 0
270 IF LEN(B$))=N THEN LET N2 = ASC
    (MID$(B$,N,1))
280 IF N1=0 THEN GOTO 500
290 IF N2=0 THEN GOTO 500
300 IF N1<>N2 THEN GOTO 500
310 LET N=N+1
320 GOTO 210
```

Como en el caso anterior, es conveniente probar este bloque de programa antes de proseguir con la codificación del resto del ejemplo. Para hacer esto, podemos recordar los ejemplos que se han estudiado en el capítulo 4, comparación de ABETO con CODO, CODO con CODO y GARCIA con GARZA. Para seguir el buen funcionamiento de nuestro programa, podemos añadir una instrucción PRINT en la línea 305, que imprima los valores de N, N1 y N2, de tal manera que podamos ver la evolución de estas variables y comprobar si el funcionamiento es correcto. Deberíamos entonces escribir:

```
305 PRINT N,N1,N2
```

También podríamos escribir en la línea 500 una instrucción parecida:

```
500 PRINT "FINAL ";N,N1,N2
```

Estas instrucciones sirven únicamente para probar el programa, cuando sepamos que funciona, se pueden quitar.

Una vez escritas, procedemos a las pruebas correspondientes. Tecleamos RUN, en la primera entrada escribamos ABETO, y en la segunda CODO, en el operador relacional introduzcamos uno cualquiera, por ejemplo, el signo igual «=». Este no tiene ningún efecto, de momento, pues nuestro programa aún no está completo.

En la pantalla veremos aparecer a continuación:

```
FINAL 1        65        67
```

que nos indica que ha encontrado la diferencia en la primera letra (por esto

escribe el 1), y que corresponde a las letras con número de orden 65 (la A), y 67 (la C).

Tecleemos nuevamente RUN, e introduzcamos CODO como primer y segundo textos; en el operador podemos continuar con el signo igual.

En pantalla aparecerá:

1	67	67	(Código ASCII de C)
2	79	79	(Código ASCII de O)
3	68	68	(Código ASCII de D)
4	79	79	(Código ASCII de O)
FINAL	5	0	0

La última instrucción nos indica que en la quinta letra hemos terminado con los textos, y el 0, indica que ya se han acabado, es decir, que ambos textos tenían cuatro letras.

Finalmente ejecutamos el programa con los textos GARCIA y GARZA, con el operador que deseemos. Observaremos en pantalla la lista siguiente:

1	71	71			G	G
2	65	65			A	A
3	82	82			R	R
4	FINAL	4	67	90	C	Z

Después de estas pruebas, ya podemos eliminar las instrucciones 305 y 500 y pasar a escribir la última parte del programa.

Otra estructura de cálculos de alternativa múliple

Será necesario establecer unas operaciones distintas según el operador solicitado, es decir debemos calcular el resultado según el operador que deseemos. La codificación debe basarse en un cálculo de las seis alternativas, antes de efectuar la operación propiamente dicha. Intente escribirlo antes de consultar la solución.

La forma de escribir estas alternativas es:

```
499 REM - Fase de calculo del resultado -
500 IF O$<>"=" THEN GOTO 530
510       LET R = N1=N2  :          REM Operador solicitado =
520       GOTO 670
530 IF O$<>"<>" THEN GOTO 560
540       LET R = N1<>N2 :          REM Operador solicitado <>
550       GOTO 670
560 IF O$<>"<=" THEN GOTO 590
570       LET R= N1<=N2 :           REM Operador solicitado <=
580       GOTO 670
590 IF O$<>"<" THEN GOTO 620
600       LET R = N1<N2 :           REM Operador solicitado <
610       GOTO 670
620 IF O$<>">=" THEN GOTO 650
630       LET R = N1>=N2 :          REM Operador solicitado >=
640       GOTO 670
650 IF O$<>">" THEN GOTO 670
660       LET R = N1>N2 :            REM Operador solicitado >
670 PRINT "El resultado es: ";R
```

Observe bien la estructura, los IF de las instrucciones 500, 530, 560, 590, 620 y 650 bifurcan a otra instrucción si no es el operador solicitado. A continuación del IF, se hace la operación correspondiente a cada operador, y se recoge en el punto 670, en donde se imprime el resultado. El REM de las líneas 510, 540 etc. es sólo una indicación para la aclaración del programa, pero, como sabe, no tiene efecto en la ejecución del mismo.

Hay que señalar que la instrucción 650 es inútil, pues por la fase de entrada, aseguramos que los únicos operadores que pueden llegar a esta fase son alguno de los seis permitidos; por tanto, si llegamos a la instrucción 650, el operador no es el «=», ni «<=», ni «<», ni «>=», por tanto, necesariamente es «>».

También es necesario recordar, la observación que se ha hecho en el caso de la alternativa múltiple (Apartado 4 de este capítulo), cada una de las instrucciones podría haberse escrito como:

```
500 IF O$ = "=" THEN LET R = N1 = N2:
    GOTO 670
510 IF O$ = "<>" THEN LET R = N1<>N2:
    GOTO 670
```

etc. Sin embargo preferimos utilizar la fórmula más larga porque en la práctica suele ser más corriente en programas más complicadas. Sin embargo, en este caso concreto, esta manera alternativa es perfectamente válida.

Ahora ejecute el programa entrándole textos y operadores y el programa contestará 1 ó 0 según la respuesta sea SI o NO.

Finalmente, hay que decir que el programa que se ha desarrollado se puede reducir a lo siguiente:

```
NEW
 10 REM : Fase de entrada
 20 INPUT "Primer texto ",A$ : REM Entrada del primer texto
 30 INPUT "Segundo texto ",B$ : REM Entrada del segundo texto
 40 PRINT "Entre uno de los operadores siguientes: "
 50 PRINT "<, <=, <>, =, >, >="
 60 INPUT O$
 70 REM : Fase de resultado
 80  IF O$"=" THEN LET R = A$ = B$ :GOTO 300
 90  IF O$"<>" THEN LET R = A$ <> B$ :GOTO 300
100  IF O$"<=" THEN LET R = A$ <= B$ :GOTO 300
110  IF O$"<" THEN LET R = A$ < B$ :GOTO 300
120  IF O$">=" THEN LET R = A$ >= B$ :GOTO 300
130  IF O$">" THEN LET R = A$ > B$ :GOTO 300
140 PRINT "El operador que ha entrado es incorrecto"
150 PRINT "Intentelo otrá vez"
160 GOTO 40
300 PRINT "El resultado es: ";R
```

La comparación entre textos en BASIC sigue el mecanismo que se ha descrito en el programa que se ha diseñado

Se han de advertir dos hechos importantes. Primero, el resultado se obtiene por comparación directa de los textos, sin necesidad de todo el bloque de cálculo. La elección de este ejemplo se ha hecho precisamente porque este esquema de comparación es el que realiza el BASIC de manera automática cuando se comparan textos. Lo hemos utilizado para ilus-

trar qué significa orden lexicográfico, para simular en BASIC lo que la máquina realiza automáticamente y, a la vez, hemos desarrollado nuestro primer programa complejo.

Segundo, observe que la instrucción 130 no es superflua como lo era la instrucción 650 en el programa anterior. Ahora, a medida que encontramos el operador a calcular, realizamos la operación, pero a la vez, si no encontramos ninguno de los operadores que deseamos, detectamos el error y volvemos a solicitar otro.

Finalmente, observar que para la redacción del último programa se ha utilizado la forma más corta de escribir la alternativa múltiple ya que sólo hay que realizar dos instrucciones, el cálculo y el GOTO, cuando se elige una alternativa.

RESUMEN

La construcción IF...THEN...ELSE es semejante al IF..THEN pero incluye el elemento ELSE, que significa en caso contrario y contiene la instrucción a realizar en el caso de que la pregunta binaria tenga respuesta negativa.

Esta estructura tiene gran importancia en el diseño de programas como estructura para codificar acciones con dos alternativas. Repase la figura 2 que muestra la manera de realizar los bloques aun cuando el BASIC no disponga de la parte ELSE.

Las preguntas con muchas alternativas deben plantearse de una manera sistemática de tal forma que las acciones se agrupen en bloques detrás de la pregunta correspondiente. Esta pregunta inicia el bloque y se hace en forma negativa para realizar el salto si no se trata de la alternativa deseada.

Por otra parte cada bloque termina con un salto al final de la construcción de todas las alternativas.

En la realización de un programa se distinguen tres fases:

La primera es la definición del problema, que consiste en escribir el enunciado y la resolución del problema de una manera más precisa que el lenguaje natural.

La segunda es la codificación del problema que consiste en pasar las ideas de la fase de definición a un lenguaje de programación.

La tercera es la prueba de los programas que verifican, hasta cierto punto, la parte de la codificación. Debe tenerse en cuenta que las pruebas nos demuestran los errores pero que no nos demuestran que no los hay.

La progresión en la realización de un programa consiste en ir codificando trozos pequeños y probando su funcionamiento correcto. A medida que el trozo se hace más grande más difícil es hacer pruebas que garanticen un funcionamiento correcto.

EJERCICIOS DE AUTOCOMPROBACION

Completar las frases siguientes:

26. La .... completa consiste en una pregunta binaria, una parte THEN y una parte ELSE.

27. La construcción .... sólo se encuentra en algunos BASIC.

28. La construcción IF THEN ELSE se utiliza frecuentemente en el .... de programas.

29. Se puede .... la construcción IF THEN ELSE en los dialectos del BASIC que no la permiten mediante una disciplina de estructuración en bloques.

30. Las sentencias que simulan la parte THEN o la parte ELSE deben terminar con un .... a un punto común que es el final de la estructura.

31. El caso de muchas alternativas se resuelve reuniendo en bloques de instrucciones que se inician por un IF que pregunta si no es la alternativa que sigue, en este caso se salta al .... de otro bloque.

32. Los bloques de instrucciones en el caso de la alternativa múltiple acaban con un salto a un punto ...., que es el final de la estructura.

33. La fase de realización de un programa en el que se prueba la codificación se denomina fase de .... y ....

34. La fase de definición de un problema consiste en describir su solución con un lenguaje más .... que el de programación.

35. La prueba de un programa detecta los errores, pero nunca demuestra la .... del mismo.

Encierre en un círculo la respuesta que corresponde a la alternativa correcta.

36. En la mayoría de dialectos BASIC que permiten la instrucción IF... THEN... ELSE, detrás del THEN y del ELSE puede haber:

    a) Una sentencia.

    b) Una sola instrucción.

    c) Sólo la instrucción GOTO.

    d) Sólo la instrucción REM.

37. El planteamiento sistemático de una decisión con varias alternativas consiste en:

    a) Agrupar todas las preguntas al inicio de todo.

    b) Colocar las preguntas al final de todo.

    c) Desdoblar en bloques que satisfagan cada una de las alternativas.

    d) Agrupar alternativas parecidas en una sola pregunta.

38. La fase de diseño consiste en definir:

    a) Los datos del problema.

    b) Los resultados del problema.

    c) Los cálculos del problema.

    d) Todas las anteriores.

39. En la fase de pruebas hay que escoger trozos de programas:

    a) Muy largos, se prueba todo a la vez.

    b) Muy cortos.

    c) Trozos que realicen funciones definidas y completas.

    d) No hay que probar los programas.

40. Las pruebas sirven para:

    a) Detectar errores en el programa.

    b) Demostrar que un programa es correcto.

    c) Para averiguar qué sentencias están mal escritas.

    d) Comprobar el buen funcionamiento del ordenador.



Capítulo 6
• Operadores lógicos.

ESQUEMA DE CONTENIDO

6.0 OBJETIVOS

En este capítulo se estudian las operaciones que podemos realizar en el ordenador mediante variables de significado lógico, es decir variables booleanas.

Es muy importante conocer la manera cómo el ordenador manipula las variables lógicas ya que es necesario disponer de estas operaciones para el proceso de toma de decisiones.

Este capítulo se divide en dos partes, en la primera se establecen las propiedades matemáticas de los operadores y se estudia con todo detalle su comportamiento.

En la segunda, se relacionan estas propiedades matemáticas con nuestra manera de formular las cuestiones lógicas, se señalan las diferencias que surgen en nuestra manera de pensar con las propiedades matemáticas que se han desarrollado en la primera parte.

Los operadores lógico-matemáticos que se estudian son el NOT, el AND, el OR y el XOR, que se traducen respectivamente por *NO, Y, O* y *O exclusivo.* Estos operadores constituyen la base de la teoría matemática asociada con la lógica.

La dificultad mayor en su utilización en expresiones complejas es que hay que definir unas prioridades, como cuando estudiamos los operadores aritméticos. Por ejemplo, en el caso de las operaciones de suma y multiplicación en los cálculos aritméticos.

Para estudiar su comportamiento se introducen las tablas de verdad que son la manera más fácil de estudiar las propiedades de operaciones complejas. Debe hacer un esfuerzo importante en manejarlas con agilidad.

Se introduce un apartado dedicado a las expresiones para cerrar un tema que constituye la base de cualquier lenguaje de programación y naturalmente el BASIC no es una excepción.

La visión unificada de todas las operaciones lógicas, aritméticas, textuales y relacionales de que dispone el lenguaje es necesaria para que aprenda a utilizar expresiones complejas en sus programas, ahorran memoria y tiempo de cálculo.

Todas estas propiedades matemáticas se estudian por su gran utilidad en el proceso de toma de decisiones. Sin embargo, su aplicación no es tan directa como puede parecer. En nuestra manera de pensar surgen ciertas ambigüedades que a la hora de trasladar a cálculos con el ordenador debemos resolver.

Estudiaremos en esta última parte del capítulo los procesos de conjunción lógica, unión lógica y la negación de las preguntas tal como las solemos formular para resolver un problema.

Es precisamente esta última parte la que le permitirá conectar el planteo de los problemas que se hace desde nuestro punto de vista con la solución que requiere el ordenador, para resolver el problema de manera informática. El salto de lo que es una fórmula de cálculo con lo que es un algoritmo para resolver un problema está estrechamente relacionado con todos los mecanismos lógicos que se exponen en esta lección.

De hecho, aunque no se mencione en el resto de la lección, la base primaria del funcionamiento de los ordenadores está en la aplicación de toda la lógica a los circuitos electrónicos.

Aprender a razonar con precisión lógica es el objetivo global de la lección ya que el razonamiento de este tipo constituye los ladrillos sobre los que se edifica toda la ciencia informática.

En todo caso, queremos hacerle desde el principio una observación: El tema que va a estudiar es un poco árido y pesado. No queremos engañarle. Es más, la mayoría de los tratados dedicados al BASIC que pueda encontrar apenas lo tratan y normalmente se limitan a indicarlos, diciendo su significado.

Sin embargo, dada su importancia, nosotros hemos preferido tratarlos ampliamente, siendo conscientes de que de haberlos pasado por alto usted no habría notado su ausencia. Pero preferimos no privarle de esta ayuda, que si en este momento no le ve todavía su importancia, a medida que vaya profundizando, nos lo agradecerá.

Pero ya le advertimos, tómese este capítulo con calma y no tenga reparo en estudiar uno por uno cada operador lógico y después descansar. Asimílelos poco a poco, pero bien asimilados.

Si queda alguna parte de este capítulo sin asimilar plenamente, tampoco le impedirá comprender el contenido total de esta Enciclopedia.



6.1 LOS DATOS DE TIPO BOOLEANO

La utilización de los operadores relacionados nos ha introducido un nuevo tipo de resultado: el SI y el NO, que se representan de una manera numérica en el ordenador pero que responden únicamente a las dos posibilidades mencionadas, el SI y el NO. Estos resultados se pueden almacenar en forma de variables numéricas; de hecho ya hemos utilizado esta forma de variables en la lección 4 (vea, por ejemplo, el apartado 4.2.1 del tomo anterior), pero su contenido está limitado a dos casos: el NO (cero) y el SI (uno o menos uno según el ordenador).

Por lo tanto, son siempre posibles las instrucciones del tipo:

```
LET A = A$ = B$
LET A = A$ < B$
LET A = 5  < T
LET A = 6  < 5
LET A = 6 <= 5
```

Los datos booleanos se almacenarán como variables numéricas

No las escriba todavía.

En todos los casos la variable A contiene únicamente un SI o un NO según el resultado del operador relacional sea cierto o no lo sea. (Recuerde, como ya vimos en el capítulo 4, que el segundo igual (=) en una instrucción LET se interpreta siempre como operador relacional). Por ejemplo, tomando la primera instrucción podemos escribir:

```
LET A="Garcia" = "Perez": PRINT A
```

el resultado sería un cero; es decir NO, puesto que García no es igual a Pérez.

A las variables que contienen un resultado SI o NO se las denomina *variables booleanas o lógicas.* Ciertamente no se distinguen en nada de las variables numéricas; sólo el programador sabe que, por su naturaleza lógica, su contenido sólo puede ser un SI o un NO.

El hecho de que el BASIC no controle si la variable es lógica o no lo es, puede producir errores tal como mostraremos en el último apartado de esta lección.

El lenguaje BASIC tiene unos operadores para manipular estas cantidades de acuerdo con su significado lógico, antes que relacionarla con su significado numérico. El hecho de que el SI y el NO se representen mediante números es un accidente, no es la esencia. De hecho, todas las cosas que se representan en el ordenador pueden ser interpretadas como números. El lenguaje BASIC introduce diversas operaciones, que en último término son operaciones aritméticas, pero que responden a una necesidad de operar sobre cantidades que tienen un significado distinto.

Los operadores básicos NOT, OR, AND y XOR

Las operaciones que el BASIC introduce para manipular estas cantidades son cuatro: NOT, AND, OR y XOR, que se traducen al castellano como *NO, Y, O* y *O exclusivo.*

Estudiaremos en primer lugar su modo de comportamiento y dejaremos para más tarde su conexión con la toma de decisiones.

Para ello tomaremos siempre unas variables numéricas, que provienen del resultado de un operador relacional. Ya sabemos que estas variables numéricas sólo pueden tener dos valores: el SI y el NO, y estudiaremos el efecto de cada una de estas operaciones sobre el resultado de un operador relacional.

Insistimos en la importancia del hecho de que estas variables deben ser el resultado de una comparación (resultado de un operador relacional) ya que si posee un valor distinto del SI o el NO, el comportamiento de los operadores puede diferir de lo que digamos aquí.

6.1.1 El operador NOT

El NOT es un operador unario, es decir, actúa sobre una sola variable y el efecto es cambiar el valor de la misma, de tal manera que si la variable contiene un SI, el resultado será un NO, y si la variable contiene un NO, el resultado será un SI.

La regla que define el operador es:

El operador NOT cambia el valor de la variable booleana o lógica.

Veamos algunos ejemplos. Considere el programa siguiente:

```
NEW
10 LET A = 5<3      : REM Aseguramos que las variables A y B
20 LET B = 5>3      : REM   contengan un resultado de tipo
                            booleano, es decir un SI o un NO
                            Naturalmente A y B contienen un
                            NO y un SI respectivamente
```

```
30 PRINT A,B
40 LET R = NOT A
50 LET S = NOT B
60 PRINT R,S
```

Operador NOT Operador Unario

1.) En este programa se estudia el efecto del NOT sobre una variable que es SI y otra variable que es NO.

Naturalmente, las instrucciones 10 y 20 nos aseguran que A y B son variables numéricas que provienen del resultado de una pregunta binaria y por lo tanto sólo contienen un SI o un NO. En este caso concreto, A contiene un NO, pues 5 *NO* es menor que 3, y B contiene un SI, pues 5 *SI* es mayor que 3.

La instrucción 30 nos imprime A y B en pantalla, para visualizar estos valores; es decir nos escribirá un cero en el caso de A y un uno o menos uno en el caso de B.

Las instrucciones 40 y 50 calculan las variables R y S, que son la aplicación del operador NOT a las variables A y B respectivamente. La instrucción 60 imprime R y S debajo de los valores de A y de B. Así, para R escribirá un 1 o −1, puesto que en esta variable se almacena el valor de NOT A, y, por lo tanto, habrá cambiado de NO a SI. En el caso de S se escribirá el valor NO (cero) por la misma razón, al ser el valor de B SI.

Después de teclear RUN, se obtendrá en pantalla:

0 1 (Valores de A y de B)

1 0 (Valores de R (NOT A) y de S (NOT B))

(Se supone que el ordenador representa el SI con un uno.)

Para que comprenda mejor todo esto, vamos a poner un ejemplo más cercano a nosotros, aunque sea forzando un poco la gramática. Imagine que Ud. debe contestar a una pregunta con SI o NO. Si Ud. contestara «NO SI», estaría contestando NO. Por el contrario, si contestara «NO NO» estaría diciendo SI.

2.) Veamos ahora un nuevo programa, para ilustrar el hecho de que el NOT puede tomar parte en expresiones más complicadas.

```
NEW
10    INPUT "Entre un SI (S), o un NO (N):",A$
20    IF A$="S" THEN GOTO 50
30    IF A$="N" THEN GOTO 50
40    PRINT "Entrada incorrecta" : GOTO 10
50 REM ---- Calculo
60    PRINT A$="S", NOT(A$="S")
70    GO TO 10
```

En este programa se entra una variable textual A$. Si su valor no es S o N, llegamos a la línea 40, y el programa vuelve a pedirnos una nueva entrada.

Si la entrada corresponde a un valor S o a un valor N, se va a la instrucción 60 (pasando por la 50 que no tiene efecto en el programa), que es una instrucción PRINT que contiene la evaluación de dos expresiones: A$=«S» y NOT (A$=«S»).

El NOT forma parte de expresiones complejas

El resultado de la primera expresión, A$=«S» da un SI si A$ contiene una S, por lo tanto *SI* son iguales A$ y «S», y un NO si contiene una N, pues *NO* son iguales A$ y «S». La segunda expresión (NOT A$=«S») es la negación de la anterior.

Este ejemplo nos demuestra que el operador NOT se puede incluir en expresiones de tipo relacional para cambiar el resultado.

3.) El tercer ejemplo es una modificación del ejemplo anterior, sustituyendo la instrucción 60 por:

```
60    PRINT A$="S", NOT(A$="S"),
      NOT(NOT(A$="S"))
```

La tercera expresión, NOT(NOT(A$=«S»)), tiene valor *SI,* si A$ vale S, y tiene valor *NO,* si A$ vale N.

Observe que al ejecutar el programa, si a usted le da «S», la respuesta será:

A$=«S» → 1 ó −1
NOT A$=«S» → 0
NOT(NOT A$=«S» → 1

El mecanismo es que sobre el valor de A$=«S», el primer NOT cambia el resultado, y este resultado es cambiado otra vez por el segundo NOT. Estos dos cambios son equivalentes a ningún cambio, es decir, es lo mismo escribir NOT(NOT(A$=«S»)) que A$=«S».

En la figura 1 vemos un esquema de cómo va actuando el operador NOT. Partimos, en primer lugar, del supuesto que A$ vale «S» y, por lo tanto, el resultado del operador relacional de igualdad es SI. Al aplicarle el primer operador NOT le cambia el valor, con lo cual el resultado es NO. Al aplicarle otra vez el operador NOT este valor vuelve a cambiar a SI.

En la figura 2 tenemos el mismo caso, pero partiendo del supuesto de que A$ vale «N» y, por lo tanto, el resultado del operador relacional de

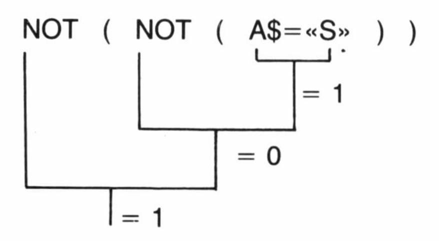


Figura 1 Doble NOT sobre un valor SI

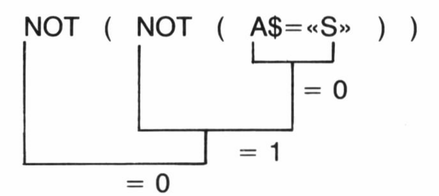


Figura 2 Doble NOT sobre un valor NO

igualdad será NO. El resultado de aplicar el primer NOT es cambiar el NO a SI. Al aplicarle el segundo, el SI cambiará otra vez a NO.

La enseñanza que debemos sacar de este ejemplo es que el NOT se puede utilizar de un modo parecido al signo menos en operaciones aritméticas; y más concretamente un doble NOT, un doble cambio de signo menos en la aritmética corriente, deja el mismo resultado inicial.

Finalmente, para expresar el mecanismo de actuación del NOT se suele utilizar lo que se llama una *tabla de verdad,* que se presenta en la figura 3.

En la primera columna se colocan las diversas alternativas posibles, en este caso una S para indicar un SI, y una N para indicar un NO. En la segunda columna, que se encabeza por un NOT, se da el resultado de aplicar el operador NOT a cada uno de los valores: es decir, al aplicar el NOT a al alternativa SI(S), el resultado es NO(N) y al aplicar el NOT a la alternativa NO(N) el resultado es SI(S).

ALTERNATIVAS	NOT
S	N
N	S

Figura 3 Tabla de verdad del operador NOT

6.1.2 El operador AND

El operador AND, que significa *y*, es un operador binario; es decir, actúa sobre dos variables lógicas, de la misma manera que los operadores binarios aritméticos actúan sobre dos cantidades aritméticas.

Le recordamos que el valor de cada una de las variables lógicas sobre las que actúa el operador AND sólo puede ser un SI o un NO. Pero antes de seguir veamos un ejemplo que nos aclare un poco más todo esto.

Imagine que desea salir de excursión, pero que para hacerlo se deben cumplir dos condiciones: Primera, que haga sol y segunda, que le hayan terminado de arreglar el coche que tiene en el taller. Vea que cada una de las dos condiciones (variables lógicas) es un SI o un NO.

Primera condición:	SI	hace sol
	NO	hace sol
Segunda condición:	SI	tengo coche
	NO	tengo coche

Figura 4 Tabla de condiciones para ir de excursión

1.ª Condición	Operador	2.ª Condición	Resultado
SI Hace sol	AND	SI tengo coche	SI excursión
SI Hace sol	AND	NO tengo coche	NO excursión
NO Hace sol	AND	SI tengo coche	No excursión
NO Hace sol	AND	NO tengo coche	NO excursión

Veamos ahora el comportamiento del operador AND, sabiendo que el resultado de aplicarlo debe ser también un SI o un NO. En la figura 4 se muestra este comportamiento.

El mecanismo que se suele utilizar para describir el comportamiento es lo que se llama *tabla de verdad* y que es muy parecida al cuadro que acaba de ver, aunque construida de un modo un poco distinta. En la figura 5 presentamos esta tabla de verdad.

La tabla tiene una forma de matriz. En la columna de la izquierda (sombreada) aparecen los valores del primer operando, y en la primera fila (también sombreada) los valores del segundo operando.

En las cuatro casillas interiores se coloca el resultado.

Se ha de observar que los resultados posibles son sólo el SI y el NO; pero tenemos cuatro posibilidades para estudiar todos los valores que se pueden dar. En efecto, para cada valor del primer operando, hay dos para el segundo operando, que dan un total de cuatro posibilidades.

Otra forma de representar esta tabla es agrupar estas columnas de posibilidades (forma más parecida al caso del NOT), del modo que se ve en la figura 6.

		Valores del segundo operando	
	AND	S	N
Valores del primer operando	S	S	N
	N	N	N

Figura 5 Tabla de verdad del operador AND

Valores de los operandos		**Resultado del AND**
1.º	**2.º**	
S	S	S
S	N	N
N	S	N
N	N	N

Figura 6 Tabla de verdad del operador AND

Operador AND Operador binario

Observe que ambas tablas dicen lo mismo aunque se dispongan de manera distinta.

Finalmente, el comportamiento del operador AND puede esquematizarse mediante la regla siguiente:

El resultado es SI sólo en el caso de que los valores de los dos operandos sea SI.

Ilustramos el comportamiento del operador AND mediante algunos ejemplos:

1.) Construcción de la tabla de verdad para el operador AND.

El programa calcula los valores de los cuatro resultados posibles de la tabla de verdad para el operador AND.

```
NEW
10   LET NO = 0
20   LET SI = NOT NO
30   LET A = SI : LET B = SI : PRINT A,B, A AND ,B
40   LET A = SI : LET B = NO : PRINT A,B, A AND B
50   LET A = NO : LET B = SI : PRINT A,B, A AND B
60   LET A = NO : LET B = NO : PRINT A,B, A AND B
```

Se consideran las cuatro posibilidades en las instrucciones 30, 40, 50 y 60, y se imprime el resultado.

Un detalle interesante es la definición de las variables NO y SI que contienen los valores del NO y del SI en la representación de su ordenador. En todos los ordenadores el cero equivale al NO, en cambio en el SI hay discrepancias; pero si utilizamos el operador NOT para cambiar el NO y obtener el SI, tenemos la ventaja de que este tipo de definición de constantes hace que un programa funcione correctamente en todos los ordenadores, independientemente del valor que utilice para representar el SI.

Al teclear el RUN aparecerá en pantalla:

```
1    1    1
1    0    0
0    1    0
0    0    0
```

(Se supone que el ordenador utiliza el 1 para representar el SI.)

2.) Este ejemplo ilustra cómo se puede utilizar el AND dentro de expresiones.

El programa que se propone pretende contestar la pregunta ¿La persona en cuestión es mujer rubia?

```
10 REM --- Fase de entrada de los operandos
20   INPUT "Entre el sexo de una persona: V para varon
         y una H para hembra ", S$
30   INPUT "Entre el color del pelo: R para rubio,
       C para castano, N para negro y O para otro tipo ", P$
40 REM --- Fase de calculo: Se intenta responder a la pregunta
50   PRINT S$="H" AND P$="R"
```

Al escribir RUN el programa nos dará un SI (1 o −1) si se cumplen ambas condiciones; es decir, si es hembra (H) y si es rubia (R).

Se podría modificar la instrucción 50 del modo siguiente para obtener una respuesta S! o NO:

```
50   LET A = S$="H" AND P$="R"
60   IF A THEN PRINT "SI"
70   IF NOT A THEN PRINT "NO"
```

El AND en expresiones

En esta nueva versión, el cálculo del operador AND se realiza en la instrucción 50 y se almacena en la variable A.

En la instrucción 60 aparece un nuevo elemento que debemos comentar. La pregunta binaria que debe haber entre el IF y el THEN se ha reducido a la variable numérica A.

Esta sustitución es posible porque la variable A tiene como contenido un valor booleano o lógico. Esta variable almacena el resultado de la comparación lógica que se ha realizado en la instrucción 50.

Se debe advertir, que es necesario que el programador controle el significado de la variable A. Si por un error introducimos cualquier número, el resultado por ahora es imprevisible; el comportamiento en este caso lo veremos en el último apartado de este capítulo.

Observe que sólo se ejecuta una de las dos instrucciones 60 o 70 pues A y NOT A no pueden ser verdad simultáneamente.

Si el ordenador dispone de la instrucción IF ... THEN ... ELSE las instrucciones 60 y 70 se pueden escribir de la manera siguiente:

```
60 IF A THEN PRINT "SI" ELSE PRINT "NO"
```

en este caso no existe la instrucción 70.

Las variables lógicas como pregunta en un IF

No es estrictamente necesario utilizar la variable A para almacenar el resultado intermedio. El IF puede contener directamente la pregunta, así la instrucción 50 puede cambiarse a:

```
50 IF S$="H" AND P$="R" THEN PRINT "SI"
   ELSE PRINT "NO"
```

en este caso no existen las instrucciones 60 y 70.

Si no se dispone de la instrucción ELSE la construcción es:

```
50 IF S$="H" AND P$="R" THEN PRINT
   "SI":GOTO 70
60 PRINT "NO"
70 END
```

6.1.3 El operador OR

Este operador es un operador binario, como lo era el AND; su modo de utilización es idéntico al AND y se diferencia únicamente respecto a éste en la tabla de verdad, es decir, en el resultado de la ejecución.

Pongamos un ejemplo para comprender el sentido de este operador OR. Imagine que las dos opciones que tiene para ir, sin mojarse, por la calle en un día de lluvia son: tomar el paraguas o ponerse el impermeable. Al igual que el caso del AND cada una de estas opciones (variables) sólo puede ser SI o NO.

Primera opción : SI tomo el paraguas
: NO tomo el paraguas
Segunda opción : SI me pongo el impermeable
: NO me pongo el impermeable

Operador OR. Operador binario

Veamos ahora también el comportamiento del OR (que significa *o*) y el resultado de aplicarlo, que lógicamente debe ser SI o NO.

En la figura 7 tiene un cuadro que muestra el funcionamiento y los resultados de aplicar el operador OR.

Veamos ahora la tabla de verdad en la figura 8 en forma de tabla cuadrada. Los valores de las variables aparecen tramados y los resultados en la parte central sin tramar.

Figura 7 Tabla de condiciones para ir sin mojarme

1.ª Condición	Operador	2.ª Condición	Resultado
SI paraguas	OR	SI impermeable	SI sin mojarme
SI paraguas	OR	NO impermeable	SI sin mojarme
NO paraguas	OR	SI impermeable	SI sin mojarme
NO paraguas	OR	NO impermeable	NO sin mojarme

Figura 8 Tabla de verdad del operador OR

		Valores del segundo operando	
	OR	S	N
Valores del primer operando	S	S	S
	N	S	N

Figura 9 Tabla de verdad del operador OR

Valores de los operandos		Resultado del OR
1.º	**2.º**	
S	S	S
S	N	S
N	S	S
N	N	N

La regla que define este operador OR es:

La respuesta es NO si ambas variables booleanas son NO, en los demás casos el resultado es SI.

Esta regla es inversa a la regla que define el AND, en el sentido que el NO sustituye al SI y el SI sustituye al NO, en la definición de la regla.

También en este caso es posible construir la tabla de verdad del modo que se ve en la figura 9.

Veamos algunos ejemplos de utilización del OR:

1.) Construyamos la tabla de verdad:

```
NEW
10 LET NO = 0 : LET SI = NOT NO
20 LET A = SI : LET B = SI : PRINT A,B, A OR B
30 LET A = SI : LET B = NO : PRINT A,B, A OR B
40 LET A = NO : LET B = SI : PRINT A,B, A OR B
50 LET A = NO : LET B = NO : PRINT A,B, A OR B
```

Observe esta vez que el comportamiento del operador se puede describir con cuatro posibilidades, pues para cada valor de la primera variable (SI o NO) tenemos valores para la segunda, lo que supone un total de cuatro valores.

2.) En este ejemplo se muestra como el operador OR participa en las expresiones con las variables booleanas. Se trata de saber si un número está dentro del intervalo de 5 a 20 ambos inclusive.

```
NEW
10   INPUT "Entre un numero ",A
20    LET C = A<5 OR A>20
30    IF C THEN GOTO 60
40        PRINT "El numero esta en el intervalo de 5 a 20"
50        GOTO 10
60        PRINT "El numero esta fuera del intervalo de 5 a 20"
70        GOTO 10
```

Tenga en cuenta que, como en los ejemplos vistos anteriormente A<5 y A>20 dan como resultado una variable de tipo booleano que permite la conjunción mediante el operador OR.

Es necesario comentar el mecanismo de funcionamiento de la expresión A<5 OR A>20. El primer valor booleano responde a la pregunta A<5 (¿ Es A menor que 5?) y el segundo valor booleano responde a la pregunta A>20; (¿Es A mayor que 20?) entonces, sólo la respuesta NO a la primera y la respuesta NO a la segunda dan un resultado NO.

Vamos a considerar algunos casos con más detalle. Para ello seguiremos el programa mediante la tabla siguiente:

A	A<5	A>20	OR
3	S	N	S
6	N	N	N
25	N	S	S

El OR también participa en expresiones complejas

La primera columna indica el número entrado, la segunda columna indica el resultado de la pregunta binaria A<5, la tercera columna da el resultado de la pregunta binaria A>20 y finalmente, la cuarta columna indica el OR de las dos columnas anteriores, que corresponderá al contenido de la variable C.

Como se observa, sólo se obtienen dos NO cuando el número está comprendido en el intervalo 5 y 20. En los otros casos una de las dos preguntas es SI, y por lo tanto, el resultado es SI.

En este ejemplo, es imposible que el resultado de las dos preguntas binarias sea SI, ya que un número no puede ser simultáneamente menor que 5 y mayor que 20.

También podemos escribir el programa sin necesidad de la variable C intermedia, del modo siguiente:

```
NEW
10 INPUT "Entre un numero:",A
20 IF A<5 OR A>20 THEN GOTO 50
30    PRINT "El numero esta en el intervalo de 5 a 20"
40    GOTO 10
50    PRINT "El numero esta fuera del intervalo de 5 a 20"
60    GOTO 10
```

6.1.4 El operador XOR

También este operador es binario, y su utilización es muy semejante a la del OR y el AND. La diferencia es el resultado de la operación, o la tabla de verdad.

Para que comprenda mejor el funcionamiento de este operador XOR, que en castellano sería *o exclusivo,* más por su función que por su traducción directa, vamos a imaginar un juego que consiste en extraer dos bolas, de una en una, de una bolsa que está llena de bolas blancas y negras (para no olvidar que los valores de las variables sólo pueden ser SI o NO, formularemos, la pregunta ¿Es la bola blanca?; cuando lo sea, la respuesta será SI, cuando sea negra, la respuesta será NO). El juego consiste en que cada participante saque las dos bolas, si las saca de distinto color sigue

Figura 10 Tabla de condiciones para seguir buscando bolas

1.ª Bola	Operador	2.ª Bola	Resultado
Blanca (SI)	XOR	Blanca (SI)	NO saca más
Blanca (SI)	XOR	Negra (NO)	SI saca más
Negra (NO)	XOR	Blanca (SI)	SI saca más
Negra (NO)	XOR	Negra (NO)	NO saca más

Figura 11 Tabla de verdad del operador XOR

	XOR	Valores del segundo operando	
		S	N
Valores del primer operando	S	N	S
	N	S	N

Figura 12 Tabla de verdad del operador XOR

Valores de los operandos		Resultado del XOR
1.°	2.°	
S	S	N
S	N	S
N	S	S
N	N	N

sacando bolas; si no, cede el turno al compañero. Gana el juego el que al final obtenga más bolas.

El punto clave del juego consiste en saber cuándo un jugador debe volver a sacar o no bolas. La figura 10 presenta el cuadro de las situaciones que se pueden presentar y el resultado de las mismas.

Operador XOR. Operador Binario

La tabla de verdad que define el comportamiento del XOR es la que tiene en la figura 11.

La regla que define esta tabla de verdad es:

Da como resultado SI, si los valores son distintos y NO cuando son iguales.

Esta tabla de verdad se puede expresar en forma de tabla horizontal como se ve en la figura 12.

El programa siguiente construye la tabla de verdad del operador XOR:

```
NEW
10   LET NO = 0 : LET SI = NOT NO
20   LET A = SI : LET B = SI : PRINT A,B, A XOR B
30   LET A = SI : LET B = NO : PRINT A,B, A XOR B
40   LET A = NO : LET B = SI : PRINT A,B, A XOR B
50   LET A = NO : LET B = NO : PRINT A,B, A XOR B
```

6.2 LAS EXPRESIONES CON OPERADORES BOOLEANOS

Resumiendo, mediante las tablas de verdad, los comportamientos de todos los operadores, obtendremos el cuadro que se muestra en la figura 13.

En las columnas de los valores booleanos aparecen las cuatro combinaciones posibles de valores. A la derecha de estas columnas aparecen los resultados de los operadores binarios AND, OR y XOR. A la izquierda, y refiriéndose únicamente a la columna del primer valor booleano aparecen los resultados del operador unario NOT.

Debemos considerar la expresión de cálculos complejos con estos operadores. El primer elemento a considerar es la prioridad de los operadores. Debe recordar el caso mencionado en la lección segunda sobre la evaluación 3+4*5 y (3+4)*5. En efecto, consideramos la operación:

A OR B AND C

Si consideramos los posibles resultados, veremos que no es lo mismo (A OR B) AND C que A OR (B AND C). En efecto, consideremos las ocho posibilidades para las variables A, B y C.

Las ocho posibilidades provienen de que la variable A puede tener 2 valores, la variable B para cada valor de A tiene otros dos valores posibles (por tanto ya son cuatro para la combinación de los valores de A y B), para cada una de estas combinaciones posibles existen 2 valores posibles de la variable C, en total pues, son ocho posibilidades.

La figura 14 muestra la tabla de los cálculos. Las tres primeras columnas indican los posibles valores de A, B y C, con el fin de construir sistemáticamente todas las combinaciones de las ocho filas, que sabemos que corresponden a todas las posibilidades. Hay que empezar colocando la columna de A: en la mitad de las filas se coloca un SI, y en la otra mitad un NO. En la columna B hay que colocar dos SI y dos NO alternativos. Finalmente, en la C hay que colocar alternativamente un SI y un NO. Observe que la progresión de colocación de SI y NO es: en A cuatro SI y cuatro NO (la mitad de ocho). En la segunda, la progresión es de dos SI y dos NO (la mitad de cuatro), y en la tercera la progresión es de un SI y un NO (la mitad de dos).

Unario	**Valores Booleanos**		**Operadores Binarios**		
NOT	1.°	2.°	AND	OR	XOR
	S	S	S	S	N
N	S	N	N	S	S
	N	S	N	S	S
S	N	N	N	N	N

Figura 13 Resumen de las tablas de verdad de los operadores lógicos

Figura 14 Tabla de verdad para el cálculo de (A OR B) AND C y A OR (B AND C)

1 2 3 A B C	4 A OR B	5 (A OR B) AND C	6 B AND C	7 A OR (B AND C)
S S S	S	S	S	S
S S N	S	N	N	S
S N S	S	S	N	S
S N N	S	N	N	S
N S S	S	S	S	S
N S N	S	N	N	N
N N S	N	N	N	N
N N N	N	N	N	N

La columna 4 es el cálculo de A OR B. Se toman las columnas A y B, y si en algunas de ellas hay un SI, en esta columna hay que colocar un SI.

En la columna cinco se muestra el cálculo de (A OR B) AND C, es decir, se aplica el operador AND entre las columnas tercera y cuarta, y se coloca un SI en la fila donde haya un SI simultáneamente en la tercera y cuarta columnas.

La sexta columna da el resultado de B AND C, mediante la colocación de un SI en todas las filas que tengan un SI en la segunda columna (correspondiente a B), y simultáneamente en la tercera (correspondiente a C).

La columna siete es el resultado de A OR (B AND C), que consiste en colocar un NO en aquellas filas que tengan un NO en las columas primera y sexta.

Es interesante que una vez explicado el mecanismo de construcción, Ud. mismo tome un papel y un lápiz e intente reproducir esta tabla sin mirar la ya construida. Una vez finalizada compare las tablas obtenidas, intente razonar las diferencias si las hay.

La prioridad en las operaciones lógicas es necesaria

Si compara ahora el resultado de (A OR B) AND C (columna quinta de la tabla) y el de A OR (B AND C) (columna séptima), verá que los resultados son distintos, por lo tanto, A OR B AND C es una expresión que puede significar varias cosas, y, por lo tanto, ambigüa si no definimos alguna regla para deshacer la ambigüedad.

Recuerde que esta situación no es nueva; en el capítulo 3 se hablaba de la prioridad de los operadores aritméticos, y decíamos que la expresión:

3+4*5

era equivalente a

3+(4*5)

pues el operador de multiplicación es prioritario al operador suma. En cambio, si queremos calcular (3+4)*5 es obligatorio colocar los paréntesis para saltarse la prioridad.

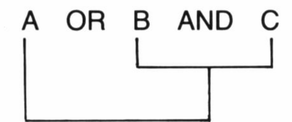
Esta misma situación se repite en los operadores booleanos. La tabla de prioridades es:

Operador	Prioridad
NOT	4
AND	3
OR	2
XOR	1

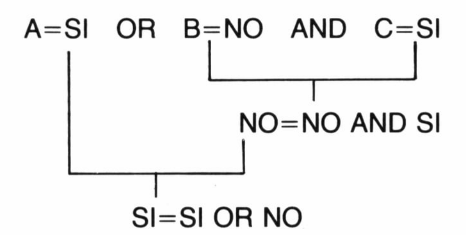
Es decir, en una expresión primero se evalúan los NOT, luego los AND, luego los OR y luego los XOR.

En el ejemplo que considerábamos A OR B AND C, debido a las reglas de prioridad es equivalente a A OR (B AND C), es decir el resultado correspondiente a la columna séptima.

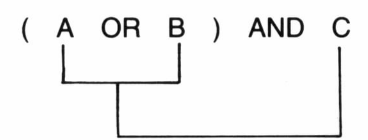
Expresado en forma de control de evaluación:



Por ejemplo, si A vale SI, B vale NO y C vale SI, el árbol de evaluación será:



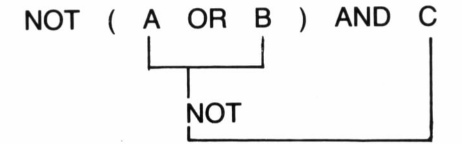
Como en el caso de los operadores aritméticos el paréntesis sirve para obligar las prioridades, de tal manera que para (A OR B) AND C, el esquema será:



Veamos algunos ejemplos de otras expresiones más complicadas, como por ejemplo:

NOT (A OR B) AND C

el orden de evaluación es:

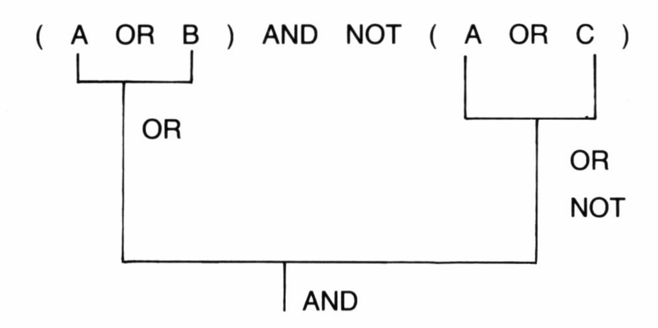


Las expresiones se evalúan de izquierda a derecha si la prioridad de los operadores es la misma

Es decir, primero se evalúa A OR B porque está encerrado entre paréntesis. A continuación se le aplica el NOT porque el operador es de más alta prioridad que el AND, y finalmente este resultado se combina con C a través del AND. Recuerde siempre, en las evaluaciones de este tipo, que el operador intenta evaluar la expresión de izquierda a derecha. Esto se ilustra en el ejemplo siguiente:

(A OR B) AND NOT (A OR C)

el orden de evaluación es:



Primero se evalúa el primer paréntesis, después se intenta evaluar el AND pero debe diferirse el cálculo hasta conocer el resultado del NOT que es de más prioridad que el AND y el cálculo debe diferirse hasta que se evalúe el paréntesis de cálculo de A OR C.

Para finalizar el tema de las expresiones booleanas vamos a experimentar con un programa que pretende calcular la expresión (NOT A AND NOT B) OR (A OR B), en donde A y B son datos que calculamos a partir de entrar valores textuales S o N.

El programa es el siguiente:

```
NEW
10 INPUT "Primer operando:", A$
20 IF A$<>"S" AND A$<>"N" THEN GO TO 10
30 INPUT "Segundo operando:",B$
40 IF B$<>"S" AND B$<>"N" THEN GO TO 30
100 REM----------------------------------
110 REM Calculo propiamente dicho
120 LET A = A$="S" : LET B= B$="S"
130 LET C = (NOT A AND NOT B) OR ( A AND B)
140 PRINT A;TAB(10);B;TAB(20);C
150 GO TO 10
```

El comportamiento del programa consiste en:

1.) Las instrucciones 10 y 20 filtran la entrada de datos de tal manera que sólo son aceptables los textos que son S o N.

2.) Las instrucciones 30 y 40 realizan también la misión de filtro para el texto B$.

3.) La instrucción 120, calcula las variables lógicas A y B, según el valor del texto sea S o N.

4.) La instrucción 130, calcula la expresión compleja y la coloca en la variante lógica C.

5.) La instrucción 140 imprime los resultados.

6.) La instrucción 150 envía al cálculo de una nueva combinación.

Si ahora hace ejecutar el programa con la instrucción RUN, podemos realizar los cálculos siguientes:

a) Introduzca un SI para el primer y segundo operando, el resultado debe ser SI.

b) Introduzca un NO para el primer y segundo operando, el resultado debe ser SI.

c) Introduzca un SI para A y un NO para B, el resultado debe ser NO.

d) Introduzca un NO para A y un SI para B, el resultado debe ser NO.

Si analizamos un poco el tipo de cálculo que hemos realizado, se puede constatar que se han realizado las cuatro posibilidades de resultado de la expresión. Efectivamente esta expresión sólo depende de dos variables A y B aunque combinadas de manera compleja. Por lo tanto, se han obtenido todos los resultados posibles para cualquier combinación de los datos de entrada.

Este tipo de cálculo se denomina la realización de la tabla de verdad de la expresión, es decir, se pueden resumir en forma de tabla todos los resultados de esta expresión del mismo modo que lo hacíamos para los resultados de los operadores AND, XOR y OR.

En este caso la tabla queda como:

A	**B**	**(NOT A AND NOT B) OR (A AND B)**
S	S	S
S	N	N
N	S	N
N	N	S

El programa nos permite recoger todos los resultados posibles sin necesidad de realizar todos los cálculos cada vez.

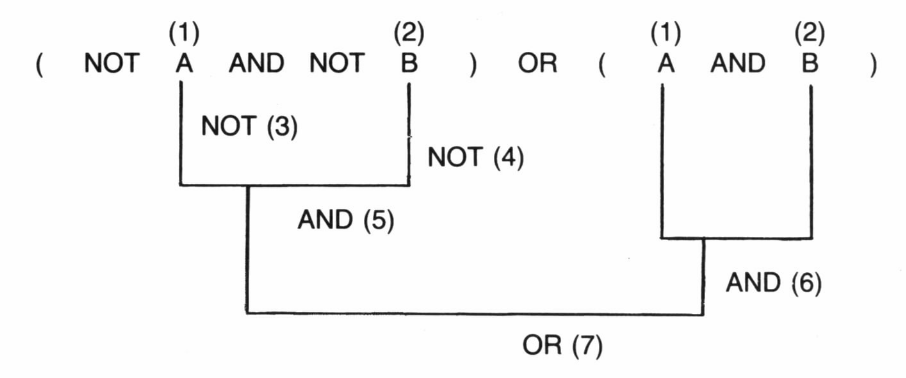
Tabla de verdad de una expresión lógica

Esta sistemática de análisis de problemas lógicos es una técnica corriente y se denomina construcción de la tabla de verdad.

Aunque la tabla de verdad resume bien el comportamiento de cálculo de una expresión, no nos indica la manera cómo se realiza el cálculo.

Como estamos estudiando el cálculo de expresiones complejas, vamos a describir cuales son las operaciones que se realizan mediante una tabla que describirá paso a paso este cálculo. La figura 15 contiene el esquema de cálculo.

Las dos primeras columnas contienen los valores de A y de B. En las columnas siguientes se van colocando los resultados intermedios según el siguiente diagrama de evaluación:



El diagrama de evaluación va acompañado de una numeración que indica el orden en que se ejecutan los cálculos y se corresponden directamente con las columnas de la figura 15.

El primer operador que se encuentra es el NOT aplicado a la variable A, luego la tercera columna contiene, el resultado de NOT A.

A continuación se encuentra el operador NOT sobre la variable B, y, por lo tanto, la cuarta columna contiene el NOT B.

El operador AND aplicado a las columnas 3 y 4 se coloca en la columna quinta que va rotulada con un AND únicamente.

A continuación se debe realizar el AND entre las variables A y B, esto se realiza en la columna sexta.

Finalmente hay que calcular el OR entre los dos resultados intermedios finales, es decir, entre las columnas quinta y sexta que darán la columna séptima.

Dos expresiones con tablas de verdad iguales son idénticas

Naturalmente después de todas estas operaciones, llegamos al mismo resultado que el que hemos obtenido con el programa, pero con una descripción del mecanismo muy detallada.

1 A	2 B	3 NOT A	4 NOT B	5 AND	6 AND	7 OR
S	S	N	N	N	S	S
S	N	N	S	N	N	N
N	S	S	N	N	N	N
N	N	S	S	S	N	S

Figura 15 Tabla de verdad para la expresión (NOT A AND NOT B) OR (A AND B)

En resumen el comportamiento de una expresión compleja con variables lógicas se puede establecer mediante una tabla de verdad. La tabla de verdad consiste en analizar el resultado que se obtiene para toda la expresión bajo cualquier combinación de las variables lógicas que intervienen. El número de combinaciones es multiplicar tantas veces dos como variables tengamos, así, si tenemos dos variables habrá 2 × 2 cuatro combinaciones, si tenemos tres habrá 2 × 2 × 2 ocho combinaciones.

Para construir de una manera sistemática estas combinaciones, tomaremos la primera variable y colocaremos la primera mitad con SI y la segunda mitad con NO, la mitad de SI la dividiremos en dos y colocaremos la mitad superior con un SI para la segunda variable y la mitad NO para la segunda variable. Así subdivideremos sucesivamente cada una de las variables siguientes, hasta llegar a la última. En ella necesariamente debe aparecer un SI y un NO alternados.

Para evaluar la tabla de verdad se hace el diagrama de evaluación y se establece el orden en que deben realizarse las operaciones, una vez establecido se van colocando columnas adjuntas a las de las combinaciones y se va calculando el resultado.

6.3 LAS EXPRESIONES CON EL ORDENADOR

En este momento ya hemos visto todos los operadores que constituyen la gama que dispone el BASIC para el cálculo de las expresiones. Es el momento de hacer un repaso de todos en conjunto, pues las expresiones son el puntal sobre el que se asienta toda la programación.

Se distinguen cuatro tipos de operadores según el tipo de operaciones que realizan:

1.) Operadores aritméticos: Realizan operaciones aritméticas entre variables numéricas. Se han estudiado extensamente en el capítulo 3. Los operadores que se clasifican en este tipo son: el cambio de signo (− unario), el mantener el signo (+ unario), la suma (+), la resta (−), la multiplicación (*), la división (/) y la potenciación (↑).

2.) Operadores textuales: Realizan operaciones sobre variables textuales. También se han estudiado en el capítulo 3. El operador que se clasifica en este tipo es la unión de textos o concatenación (+).

3.) Los operadores relacionales: Realizan operaciones de comparación entre variables numéricas o variables textuales, su resultado es siempre un valor booleano, es decir, un valor númerico SI o un valor numérico NO. Se han estudiado en el capítulo 4. Se clasifican en este tipo de operadores los de igualdad (=), desigualdad (<>), mayor que (>), mayor o igual que (>=), menor que (<) y menor o igual que (<=).

4.) Operadores lógicos: Realizan operaciones entre variables booleanas, dando como resultado un valor booleano, es decir un valor numérico que sólo puede ser SI o NO. Se clasifican en este tipo de operadores, el NOT, el AND, el OR y el XOR.

Todos los operadores pueden mezclarse en una operación compleja

En una expresión pueden aparecer todos los tipos de operadores mezclados, por ejemplo, en la expresión siguiente:

```
A+5 < B AND A$="S"+B$
```

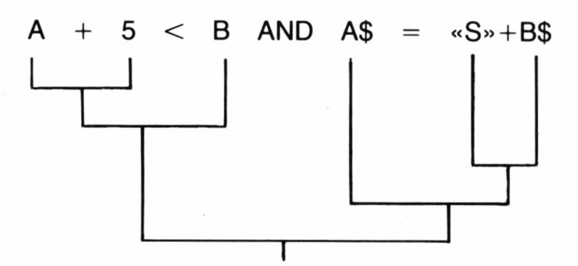
aparecen los operadores, + aritmético en A+5, el operador + textual en «S»+B$,

Los operadores relacionales < en A+5<B y el = en A$ = «S»+B$ y también el operador lógico AND.

La evaluación de esta expresión sigue también unas normas de jerarquía o prioridad de tal manera que la orden de evaluación se establece mediante esta prioridad de una manera precisa.

La figura 16, establece la tabla de prioridades para todos los operadores en conjunto, en el ejemplo que se ha propuesto el orden de evaluación sería el siguiente:

A + 5 < B AND A$ = «S»+B$



Como puede observar primero se intentan resolver los operadores aritméticos y textuales, después los operadores relacionales y finalmente los lógicos.

Cuando se quiere modificar el orden de ejecución determinado por esta tabla de prioridades se utilizan los paréntesis del mismo modo que se ha explicado en las operaciones aritméticas.

Observando estas normas se dará cuenta de que en muchos ejemplos de los utilizados en este capítulo se pueden acortar si se utiliza el cálculo general de expresiones. No se hace por dos razones: la primera para poner de manifiesto las variables lógicas implicadas y en segundo lugar para no tener problemas con las prioridades.

Si consideramos de nuevo el programa de ejemplo para el cálculo de expresiones lógicas complicadas que da origen a la figura 15, se puede escribir del modo siguiente:

```
10 INPUT "Primer operando:",A$
20 IF A$<>"S" AND A$<>"N" THEN GO TO 10
30 INPUT "Segundo operando:",B$
40 IF B$<>"S" AND B$<>"N" THEN GO TO 30
50 IF NOT A$="S" AND NOT B$ = "S" OR A$="S" AND B$="S"
          THEN PRINT "SI" : GO TO 10
60 PRINT "NO"
70 GO TO 10
```

Figura 16 Tabla general de prioridades para los operadores del lenguaje de programación BASIC

Tipo	Operador	Prioridad
Aritméticos	Cambio de signo (−) Mantener signo (+) Funciones	10
	Potenciación (↑)	9
	Multiplicación (*) División (/)	8
	Suma (+) Resta (−)	7
Textuales	Unión de textos (+)	6
Relacionales	Igualdad (=) Desigualdad (<>) Mayor que (>) Mayor o igual que (>=) Menor que (<) Menor o igual que (<=)	5
Lógicos	Negación NOT	4
	Conjunción AND	3
	Unión OR	2
	Exclusión XOR	1

En la instrucción 20 la expresión en el interior del IF previene de la posibilidad de que A$ sea distinto de S o N. Exactamente la misma función realiza la instrucción 40.

En efecto, si A$ es una A, por ejemplo, es verdad que A$ es distinto de S y también distinto de N con lo que se realiza la parte del THEN que consiste en enviar a la instrucción 10 que realiza de nuevo la pregunta.

Los paréntesis alteran las prioridades de ejecución

Por otra parte, si A$ es S la primera parte de la pregunta binaria es falsa, y entonces ya no se ejecuta la parte asociada del THEN porque para que se realice es necesario que las dos subpreguntas que se realizan sean verdaderas simultáneamente, que es lo que requiere el operador AND. Si A$ es igual a N, entonces la parte que no es verdadera es la segunda subpregunta.

En cuanto a la pregunta en la instrucción 50 es el cálculo de la expresión en la que se han suprimido las variables lógicas intermedias, que nos expresan los resultados de las preguntas A$=«S» y B$=«S».

En conclusión, el BASIC permite en las expresiones la introducción de todos los operadores y funciones en el cálculo de expresiones complejas. Este tipo de operaciones suelen ahorrar memoria y tiempo de cálculo. Las prioridades de la ejecución de los operadores se realiza según la tabla de la figura 16. Cuando se quiere modificar el orden de ejecución se utilizan los paréntesis.

RESUMEN

En BASIC el resultado de una operación lógica SI o NO se puede almacenar en una variable numérica que llamamos variable booleana o lógica.

Desde el punto de vista de escritura de las variables una variable lógica no se distingue de una variable numérica. Por lo tanto, es responsabilidad del programador saber cuáles son las variables numéricas que tienen significado lógico.

El BASIC tiene cuatro operadores para manipular las variables lógicas que son: el NOT, que es unario, y los tres operadores binarios AND, OR y XOR.

El comportamiento del operador NOT consiste en cambiar el valor NO a SI y el valor SI a NO.

El comportamiento del operador AND es dar como resultado SI si el valor de ambos operandos es SI; en los demás casos el resultado es NO.

El comportamiento del operador OR es dar como resultado NO si el valor de ambos operandos es NO; en los demás casos el resultado es SI.

El comportamiento del operador XOR da como resultado SI si el valor de ambos operandos es distinto y NO si ambos operandos tienen valores iguales.

La manera de expresar los resultados de los operadores y de las expresiones lógicas en general es mediante las tablas de la verdad.

La tabla de verdad consiste en varias columnas para indicar el valor de los operandos y una columna para indicar el resultado.

El número de filas que tiene la tabla de verdad depende del número de operandos, si tenemos 1 operando sólo hay dos filas (el caso del NOT), si tenemos 2 operandos hay 4 filas, 2 × 2 (caso del AND, OR y XOR), si tenemos 3 operandos hay 8 filas, 2 × 2 × 2 (caso de las expresiones más complejas) y así sucesivamente.

En cada fila hay una combinación de S y N para cada operando distinta. La manera de construirla consiste en tomar la primera columna y colocar la mitad de las filas a SI y la otra mitad a NO. En la segunda columna, se debe considerar por separado las filas que contienen un SI en la primera columna y las que tienen un NO. Dividiremos en la mitad SI y en la mitad NO cada una de las mitades mencionadas. Este procedimiento se repite hasta alcanzar la última columna en la que necesariamente debemos colocar un SI y un NO alternativamente.

Con los operadores booleanos podemos formar expresiones complicadas, pero en este caso es necesario, conocer la jerarquía de los operadores, pues no es lo mismo A AND (B OR C) que (A AND B) OR C.

El orden de prioridad se establece dando al NOT la máxima después el AND, luego el OR y finalmente el XOR.

Cuando se precisa cambiar el orden de ejecución se utilizan los paréntesis.

Los operadores lógicos pueden participar además en expresiones más complejas en la que intervengan operadores aritméticos, textuales y relacionales. La figura 16 nos da los valores de las prioridades en forma global para todas las expresiones que se pueden formar en BASIC; en cualquier caso para romper estas prioridades sólo basta utilizar los paréntesis.

EJERCICIOS DE AUTOCOMPROBACION

Complete las frases siguientes:

1. Una variable .... no se distingue de una variable numérica.

2. El operador NOT es ....

3. El operador NOT .... el valor de las variables lógicas.

4. Los operadores AND, OR y XOR son ....

5. El resultado del operador AND es cierto cuando las dos variables son ....

6. El resultado del operador OR es .... cuando las dos variables son falsas.

7. El resultado del operador XOR es cierto cuando las dos variables son ....

8. La .... expresa el resultado de un operador lógico y de las expresiones booleanas.

9. Los operadores .... participan de expresiones con todo tipo de operadores.

10. La prioridad de los operadores se ordena así: Primero los ...., seguidos de los textuales, los relacionales y los lógicos.

Encierre en un círculo la letra que corresponda a la alternativa correcta.

11. Una tabla de verdad de una expresión que contiene 5 variables lógicas, ¿cuántas filas tiene?

    a) 8
    b) 16
    c) 32
    d) 64

12. El resultado es cierto cuando al menos una de las dos variables booleanas lo es, corresponde al operador:

    a) NOT
    b) AND
    c) OR
    d) XOR

13. El resultado es falso cuando las dos variables son falsas o las dos variables son verdaderas, corresponde al operador:

    a) NOT
    b) AND
    c) OR
    d) XOR

14. La aplicación consecutiva de un operador lógico, es decir, aplicarlo una vez y luego aplicarlo otra vez sobre el resultado, sólo tiene sentido en el operador:

    a) NOT
    b) AND
    c) OR
    d) XOR

15. El resultado es falso cuando al menos una de las variables lógicas lo es, corresponde al operador:

    a) NOT
    b) AND
    c) OR
    d) XOR

Realizar el diagrama de evaluación de las expresiones siguientes, si A, B y C son variables lógicas (Utilice la figura 16 para establecer las prioridades).

16. NOT A AND B OR C

17. NOT A OR B AND C

18. (NOT A OR B) AND C

19. NOT (A OR B AND C)

20. A OR NOT B AND C

Realizar el diagrama de evaluación de las expresiones siguientes, en donde, las variables sin el signo dólar al final, se deben considerar numéricas (utilice la figura 16 para establecer las prioridades).

21. A < 5 − 3 AND A$ > «Juan»

22. 5 = A − 2 OR NOT A <> B

23. NOT 5 * A + B < A * A

24. A$ + B$ <= B$ + A$ OR A$ = B$

25. R$ < «Javier» OR R > 23

26. Realizar la tabla de verdad para la expresión:
    (A AND B) OR (NOT A AND NOT B)

6.4 LA TOMA DE DECISIONES

Hemos visto en el capítulo 4, que los operadores relacionales sirven para transformar las preguntas binarias sobre objetos en variables booleanas que permiten tomar decisiones simples sobre lo que deseamos realizar en nuestro programa.

En este capítulo hemos estudiado en la primera parte los operadores booleanos como operadores que tiene el ordenador y de los cuales hemos analizado múltiples propiedades que parecen más una diversión un poco pesada que algo de utilidad práctica inmediata.

En este apartado vamos a estudiar cómo se conectan estas operaciones con las preguntas que formulamos normalmente en las decisiones que se deben tomar al resolver un problema mediante un programa de ordenador.

6.4.1 Alternativas válidas

En los ejemplos que se han hecho de programas para demostrar las propiedades de los operadores ya se ha utilizado una fórmula corriente de toma de decisiones. Se trata de saber si una variable de entrada cumple unas especificaciones determinadas. En los ejemplos hemos utilizado una variable textual y queremos saber si es una S o una N pues sólo son los símbolos admitidos. La formulación de la pregunta en el lenguaje ordinario es: ¿el carácter que hemos entrado es una S o una N? En la pregunta se distinguen tres partes, la primera si el texto es una S, que es una pregunta binaria; la segunda si el texto es una N, que también es una pregunta binaria y finalmente la partícula *o* que nos une las dos preguntas binarias.

Al traducir este esquema en BASIC la solución es bastante sencilla, la primera pregunta corresponde al operador relacional A$ = «S» (se supone que el texto entrado se almacena en la variable A$), la segunda pregunta se plantea como A$=«N», la partícula que los une debe ser el OR pues es la traducción al inglés del *o* español que une las dos proposiciones.

La pregunta se formularía en el lenguaje de programación como

```
IF A$="S" OR A$ ="N" THEN...
```

Sólo hay una alternativa que es cierta

La respuesta sólo será cierta si una de las dos preguntas binarias es cierta. Si observa con atención se dará cuenta de que este tipo de preguntas no pueden tener ciertas las dos preguntas que conecta la partícula OR; la variable textual no puede ser simultáneamente «S» y «N».

Esta propiedad se da frecuentemente cuando consideramos alternativas múltiples de la misma variable, por ejemplo, si preguntamos si la variable textual puede ser una de las letras siguientes, *A, I, B, M.* Estas iniciales pueden responder, por ejemplo, a operaciones frecuentes en tratamiento de textos, referidos a las acciones de acabar, insertar, borrar, modificar.

6.4.2 Intervalos numéricos

Otro ejemplo muy frecuente de decisiones a tomar en el ordenador consiste en preguntar si una cantidad numérica está comprendida entre ciertos límites, por ejemplo, si las ventas están entre 20.000 unidades monetarias y 30.000 unidades monetarias entonces la comisión será del 15 %.

Se obtendrá una rebaja del 10 % si se compran de 15 a 20 unidades del mismo libro. Una persona puede trabajar si su edad está comprendida entre los 16 y 65 años (se considera que 65 años no permiten trabajar y 16 años sí lo permiten).

Par discutir un caso, imaginemos que nuestro programa debe tomar una acción distinta si la variable numérica A, que representa la edad de un individuo, está comprendida entre 16 y 65 años.

En primer lugar el número contenido en A debe ser mayor o igual que 16. En términos de operadores relacionales, significa que la variable A debe compararse con el operador >= con el número 16, de tal manera que obtendremos A>=16 (también es válida la fórmula 16 <= A, es decir, 16 es menor o igual que A). La respuesta a esta comparación puede ser SI ó NO como la de todos los operadores relacionales.

La conjunción de dos desigualdades da el intervalo numérico

Por otra parte, la pregunta que A es menor que 65, se concreta en el ordenador mediante el operador relacional A<65, la respuesta a esta comparación es binaria tal como corresponde a los operadores relacionales.

Sin embargo, el interés en nuestro programa no está en las dos preguntas independientes sino en la conjunción de las dos. En efecto, para que el número esté comprendido entre 16 y 65 es necesario que se formule la pregunta de una manera más precisa, tal como, ¿La edad del individuo es mayor o igual que 16 *y* la edad del individuo es menor que 65? Observe que la conjunción lógica utilizada en este tipo de pregunta es la partícula *y* colocada entre las dos afirmaciones. Al traducir esta pregunta al ordenador se sustituyen las preguntas binarias con los operadores relacionales unidos por el operador AND que es el que se corresponde con la partícula *y* de la conjunción lógica (en inglés AND significa justamente y).

La traducción de la pregunta anterior en BASIC sería:

```
IF 16<= A AND A < 65 THEN ....
```

La pregunta formulada en lenguaje ordinario sólo será cierta si se cumplen las dos preguntas binarias, se trata pues de unir los dos resultados intermedios de los operadores relacionales mediante el operador lógico que dé cierto si las dos preguntas son ciertas, se trata del operador AND, como ya se ha descrito.

El programa que decide si una persona puede trabajar después de saber su edad es el siguiente:

```
10 INPUT "Edad:",A
20 IF 16<= A AND A<65 THEN PRINT "SI":GO TO10
30 PRINT "NO": GO TO 10
```

Escriba RUN y calcule con diversas edades, por ejemplo, 12, 36, 65, 85, 10, 16, 40. Los resultados deben ser NO, SI, NO, NO, NO, SI y SI.

La utilización del operador AND le permite escribir la pregunta de una manera compacta y más directamente relacionada con la formulación de la pregunta en el lenguaje ordinario.

Se ha de notar que estrictamente hablando el operador AND no es imprescindible. El mismo programa puede escribirse del modo siguiente:

```
10 INPUT "Edad:",A
20 IF 16<= A THEN IF A<65THEN PRINT"SI":GOTO10
30 PRINT "NO": GO TO 10
```

En este caso se ha sustituido el operador AND por un doble IF, el segundo sólo se alcanza si es cierto el primero.

Se puede argumentar que es mejor la forma del segundo programa porque sólo se alcanza el segundo IF si es cierta la primera pregunta. Naturalmente este argumento es cierto, sin embargo, se gana claridad en la pregunta si se plantea con el operador AND. La diferencia de tiempos no es tan sustancial para que tenga importancia; en cambio, la relación de la pregunta con el lenguaje ordinario hace el texto del programa más claro.

6.4.3 Decisiones con dos variables

En los dos ejemplos que hemos mencionado, los operadores lógicos nos acercan a las preguntas del lenguaje ordinario; sin embargo, se han utilizado en preguntas en que sólo interviene una variable y por lo tanto, no son posibles todas las combinaciones. En el caso de la comparación con variable textual, sólo puede ser cierta una de las alternativas; en el caso del intervalo numérico, no pueden ser falsas las dos alternativas simultáneamente, pues un número no puede ser menor que 16 y mayor que 65.

Consideremos el siguiente ejemplo, una empresa que vende ciertos artículos, tiene la política de descuentos siguiente: cuando se vende a un cliente que compró más de 1.000.000 de unidades monetarias el año anterior se le hace un descuento del 10 %, si además el pedido supera las 10.000 unidades monetarias se le hace otro 10 % adicional.

Si el cliente que compra no cumple con el 1.000.000 de unidades monetarias, entonces sólo se hace un 10 % de descuento si la cantidad comprada supera las 10.000 unidades monetarias.

Se trata de hacer un programa que nos diga cuánto descuento hay que hacer sabiendo la cantidad comprada por el cliente el año anterior y la cantidad que sube el pedido que se está facturando.

Llamando QA la cantidad comprada el año anterior y QP la cantidad comprada en el pedido se puede sistematizar los descuentos del modo siguiente:

	QP>= 10000	QP < 10000
QA>=1000000	20 %	10 %
QA< 1000000	10 %	0 %

En la casilla superior izquierda se coloca un 20 % que corresponde a un 10 % por superar el 1000000 de unidades monetarias y 10 % adicional

por superar la compra las 10000 unidades monetarias. Nuestro programa, debe entrar las dos cantidades QA y QP como dato y finalmente calcular el % de descuento.

Se inicia el programa entrando las variables con el texto siguiente:

```
10 INPUT "Cantidad del pedido:",QP
20 INPUT "Cantidad del ano anterior:",QA
```

A continuación se procede a realizar las preguntas para obtener el tanto por ciento que designaremos con la variable TP.

Inicializamos la variable TP a cero, y luego formulamos las preguntas correspondientes a cada uno de los casos. En el caso del 20 % la pregunta es: ¿La cantidad QA es mayor o igual que 1000000 y la cantidad QP es mayor o igual que 10000?

En BASIC y utilizando los operadores lógicos esta pregunta se plantea del modo siguiente:

```
IF QA>= 1000000 AND QP>=10000 THEN LET TP =20
```

El operador relacional >= sobre QP nos formula la primera parte de la pregunta, el operador relacional >= sobre QP nos plantea la segunda parte de la pregunta y el operador AND sirve para conectar las dos preguntas mediante la conjunción lógica, que en lenguaje ordinario se expresa como *y*.

La parte de detrás del THEN nos da el valor de 20 para la variable TP.

La pregunta para obtener los otros casos son:

¿La variable QA no supera el millón de unidades monetarias y la variable QP supera las diez mil unidades monetarias? En BASIC esto se traduce por

```
IF QA<1000000 AND QP>= 10000 THEN LET TP = 10
```

¿La cantidad QA supera el millón y la cantidad QP no supera las diez mil unidades monetarias? Traducido al BASIC esta pregunta se formula como

```
IF QA>=1000000 AND QP<10000 THEN LET TP = 10
```

En definitiva la parte de cálculo del tanto por ciento se formula según las instrucciones siguientes:

```
100 LET TP = 0
110 IF QA>=1000000 AND QP>=10000 THEN LET TP = 20
120 IF QA<1000000 AND QP>=10000 THEN LET TP = 10
130 IF QA>= 1000000 AND QP<10000 THEN LET TP = 10
```

Preguntas exclusivas sólo hay una cierta

Es importante observar que sólo es posible que sea cierta una o ninguna de las preguntas, por lo tanto cuando salimos de este bloque de programa tendremos colocado en la variable TP el valor del tanto por ciento adecuado.

En el bloque siguiente del programa procederemos al cálculo definitivo de la cantidad que se debe facturar. El programa queda finalmente del modo siguiente:

```
10 INPUT "Cantidad del pedido:",QP
20 INPUT "Cantidad del ano anterior:",QA
100 LET TP = 0
110 IF QA>=1000000 AND QP>=10000 THEN LET TP = 20
120 IF QA<1000000 AND QP>=10000 THEN LET TP = 10
130 IF QA>= 1000000 AND QP<10000 THEN LET TP = 10
200 PRINT "Cantidad a descontar:",QP*TP/100
210 GO TO 10
```

Realice las pruebas con las cantidades siguientes:

QA = 400000 , QP = 2000

QA = 1400000, QP = 2000

QA = 100000 , QP = 15000

QA = 2000000, QP = 20000

Los resultados para la cantidad a descontar deben ser: 0, 200, 1500 y 4000.

Para finalizar hay que hacer dos observaciones:

La primera se refiere a la utilización de los operadores relacionales en las preguntas anteriores.

Por ejemplo, la pregunta

```
IF QA<1000000 AND QP>= 10000 THEN ....
```

puede formularse de la manera siguiente,

```
IF NOT QA>=1000000 AND QP>= 10000 THEN ....
```

(no hace falta colocar paréntesis pues el NOT es un operador de mayor prioridad que el AND).

El NOT sirve para negar una variable booleana. Muchas veces esta variable está implícita, es decir, no aparece definida en el programa ya que aparece como resultado de un operador relacional, en nuestro caso la variable implícita es QA <= 1000000.

Sin embargo, debido a que los operadores relacionales tienen su contrario, se puede sustituir el NOT delante de un operador relacional por el operador relacional contrario, en nuestro caso es equivalente a QA < 1000000.

Por ejemplo, negar la pregunta: ¿QA es mayor o igual a 1000000?, es lo mismo que decir ¿QA es menor que 1000000? Por lo tanto, desde el punto de vista de la programación, NOT QA >= 1000000 es exactamente lo mismo que decir QA < 1000000.

La negación de un operador relacional nos lleva al operador relacional contrario

Es decir, la aplicación del NOT sobre una pregunta formulada con un operador relacional, es equivalente a sustituir el operador relacional por su contrario. La tabla que se da a continuación describe todos los casos posibles:

```
NOT (A=B)      A<>B
NOT (A<>B)     A=B
NOT (A>B)      A<=B
NOT (A>=B)     A<B
NOT (A<B)      A>=B
NOT (A<=B)     A>B
```

Debido a la existencia de los operadores contrarios no parece necesaria la introducción del NOT. Sin embargo, muchas veces, para clarificar las preguntas que realizamos es conveniente utilizar la negación de un operador relacional antes que utilizar el operador contrario.

La segunda observación consiste en utilizar las propiedades del operador XOR. En este ejemplo en concreto, se aplica el mismo descuento tanto si se compra poco y es un buen cliente, como si se compra mucho y es un cliente esporádico, concretamente el 10 %.

La pregunta para considerar esta situación se puede realizar mediante la fórmula siguiente:

```
IF QA>=1000000 XOR QP>=10000 THEN  LET TP =10
```

en efecto, sólo será SI si una de las dos es falsa y la otra verdadera, por el contrario, será NO si ambas son verdaderas o bien ambas falsas.

El XOR se puede expresar en función del AND, el OR y el NOT

En algunos dialectos de BASIC no se dispone del operador XOR, pero puede simularse mediante una pregunta como:

(QA>=1000000 AND NOT (QP>=10000)) OR (NOT (QA>=1000000) AND QP>=10000)

Si llamamos QA >= 1000000 la variable lógica A y a QP >= 100000 la variable lógica B, la pregunta anterior se formula como:

(A AND NOT B) OR (NOT A AND B)

esta expresión es totalmente equivalente a

A XOR B

Para comprobar este resultado basta que se construya la tabla de verdad. Esta construcción se pide como ejercicio de comprobación, puede esperar a realizarla en el momento de hacer los ejercicios.

Como conclusión, cuando se plantean temas de toma de decisión en un problema que se quiere resolver mediante ordenador, es necesario, en primer lugar reducir todas las preguntas a preguntas binarias. En segundo lugar, hay que plantear la pregunta compleja mediante la utilización de operadores lógicos que encadenan las distintas preguntas binarias para formar una compleja.

6.5 PROPIEDADES DE LOS OPERADORES BOOLEANOS

Los operadores booleanos presentan una serie de propiedades que son útiles en programación y en el proceso de toma de decisiones. Vamos a describirlas sin entrar en los aspectos matemáticos.

6.5.1 El AND y el OR mediante dos preguntas sucesivas

Los primitivos lenguajes de programación de alto nivel, que surgieron cuando aparecieron los ordenadores, no disponían de operadores lógicos, aunque eran necesarios para afrontar problemas reales de decisión.

La manera de solucionar el problema consistía en encadenar instrucciones IF ... THEN de modo que el resultado fuera equivalente.

a) Caso del operador OR.

Supongamos la pregunta de si un número ESTA FUERA del intervalo numérico de 26 a 54 ambos inclusive, ya sabemos que con los operadores lógicos esto se puede formular como A<26 OR 54<A, en donde se supone que la variable A, contiene el número que queremos probar.

Un esquema alternativo sin la utilización del OR es el que se muestra a continuación: (No lo teclee en el ordenador, es sólo un esquema)

```
100 IF A<26 THEN GOTO 200
110 IF 54<A THEN GOTO 200
120 Se inician las instrucciones de la parte
    correspondiente a la respuesta negativa.
    ......
    GOTO 300
200 Se inician las instrucciones de la parte
    correspondiente a la respuesta afirmativa.
    ......
300 Continuacion del programa.
```

Efectivamente en el caso del OR, si cualquiera de las dos respuestas es afirmativa el resultado es afirmativo. Así, si A<26 es afirmativo, la instrucción 100 ya cede el control a la línea 200, pues la primera condición es verdadera. Si no es verdadera se cede el control a la línea 110. En esta instrucción si 54<A es verdadero se envía la ejecución a la línea 200, pues aunque es negativa la primera respuesta, es afirmativa la segunda; globalmente el OR es afirmativo.

Solamente se alcanza la instrucción 120 cuando las dos preguntas son negativas, es decir, el resultado del OR es negativo.

El programa completo para hacer esta prueba es:

```
10 INPUT A
100 IF A<26 THEN GOTO 200
110 IF 54<A THEN GOTO 200
120 PRINT "NO ESTA FUERA del intervalo 26-54"
130 GOTO 300
200 PRINT "SI ESTA FUERA del intervalo 25-54"
300 GOTO 10
```

Sólo el IF ... THEN es estrictamente necesario

El tipo de construcción no es más que un reflejo de la regla que dice que si una de las preguntas del OR es cierta, el resultado global es cierto.

Parece además que utilizando este tipo de construcción se gana tiempo, pues si la primera pregunta es cierta ya no es necesario realizar la segunda. Sin embargo, esto es difícil de precisar, pues se pierde tiempo cuando las dos son falsas. En general, se prefiere la utilización del OR en lugar de las dos preguntas sucesivas. Ciertamente se gana claridad en el programa y en cuanto a tiempo no es posible decidir cual es más rápida. La contestación completa a esta cuestión también depende del problema que se resuelve.

b) Caso del AND.

Las consideraciones realizadas para el OR son aplicables también al AND, aunque la forma de construcción es ligeramnete distinta. Imagine que la pregunta a contestar es O<A AND A<10. El esquema, no lo teclee todavía, es el siguiente:

```
100 IF NOT ( O<A ) THEN GOTO 200
110 IF NOT ( A<10) THEN GOTO 200
120 Se realizan las instrucciones
    que dan como afirmativa la
    pregunta O<A.y A<10
    ......
    GOTO 300
200 Se realizan las instrucciones
    que dan como negativa la
    pregunta O<A y A<10
300 Continuacion del programa
```

Compare esta construcción con la del OR y observará que es prácticamente idéntica, excepto que las preguntas se hacen en negativo; es decir, con el NOT delante. Esto es equivalente a decir que si la primera pregunta es falsa, no hace falta continuar ya que el AND precisa de las dos preguntas verdaderas para que sea globalmente verdadero. Ponga atención al considerar la primera pregunta como falsa, su negación nos dará verdadera y, por lo tanto, tomará la parte del THEN del IF en la instrucción 100. Exactamente el mismo argumento se puede utilizar para la instrucción 110. Por lo tanto, sólo se alcanzará la instrucción 120 si las dos preguntas son verdaderas y, en consecuencia, será verdadera toda la pregunta globalmente. Sabiendo además las propiedades de negación de los operadores relacionales, se puede construir el programa siguiente sin utilizar tampoco el NOT:

```
10 INPUT A
100 IF O>= A THEN GOTO 200
110 IF A>= 10 THEN GOTO 200
120 PRINT "SI"
130 GOTO 300
200 PRINT "NO"
300 GOTO 10
```

Observe que NOT (O<A) es lo mismo que O>=A y NOT (A<10) es lo mismo que A>=10.

También en este caso se aplican las consideraciones de tiempo y claridad expuestas en el caso del OR.

6.5.2 Propiedad distributiva

Otro tipo de propiedad, algo más complicada que las anteriores, es la que se llama distributividad del AND respecto del OR. La propiedad se expresa formalmente

A OR (B AND C) ≡ (A OR B) AND (A OR C)

Es decir son idénticos como indica el signo (≡) los resultados de hacer el OR entre un operando y el resultado de un AND que hacer el OR con cada uno de los operandos del AND y hacer un AND con cada resultado.

Para aclarar el significado de esta ley expresada formalmente imagine que Ud. desea comprar un coche. Considera importantes las características siguientes:

1.) Motor Diesel o de gasolina.

2.) Frenos de disco o de tambor.

3.) Tracción trasera o delantera.

Desea hacer una selección de los coches que se ofrecen en el mercado

y para que un coche merezca su consideración debe responder a la pregunta siguiente.

¿El motor es Diesel o (lleva frenos de disco y lleva tracción trasera)?

La equivalencia en los modos de preguntar

Analicemos la semejanza entre esta pregunta y la forma abstracta planteada más arriba. Las preguntas tal como están formuladas son binarias y se corresponden con las variables, así, A se corresponde con *El motor Diesel,* la B con *lleva frenos de disco* y la C con *lleva tracción trasera.* Las partículas *o* y *y* se corresponden con el OR y el AND respectivamente. El paréntesis que se ha introducido en la pregunta en lenguaje natural es sólo para resaltar esta semejanza.

La propiedad distributiva nos dice que esta misma pregunta se puede formular del modo siguiente:

¿(El motor es Diesel o lleva frenos de disco) y (El motor es Diesel y lleva tracción trasera)?

en donde, se mantiene el paralelismo con la segunda fórmula de la identidad abstracta antes mencionada.

Para mostrar (no demostrar en el sentido matemático, sino analizar intuitivamente) que las dos preguntas son idénticas hagamos el razonamiento siguiente. En la primera forma de plantearla, si el motor es Diesel ya no me preocupan el resto de propiedades, en el caso de que no sea Diesel exijo que simultáneamente tenga las dos propiedades de tracción trasera y frenos de disco.

Por lo que se refiere a la segunda forma de plantearla, se ha transformado en dos subpreguntas conectadas por un *y.* Para que el resultado sea cierto es necesario que las dos subpreguntas sean verdaderas. Cada una de las dos subpreguntas se formula con un *o.* Por lo tanto, para que la subpregunta sea globalmente cierta sólo basta con que lo sea uno de los términos.

En cada una de las subpreguntas aparece la pregunta ¿El motor es Diesel?, si ésta es cierta lo es cada una de las dos subpreguntas, y, en consecuencia, lo es la pregunta global.

En el caso de que el motor no sea Diesel, ya sólo nos importan los otros términos de las subpreguntas, lleva tracción trasera y lleva frenos de disco, conectadas por la conjunción *y*, sólo será verdad si ambas son ciertas.

Las conclusiones, como puede observar, son las mismas que la pregunta formulada de la primera forma.

De hecho para probar exhaustivamente el razonamiento será necesario ver que las tablas de verdad de una forma y otra son la misma. Lo veremos más adelante.

Un comentario aparte merece este tipo de decisión. Para elegir un coche se establece una pregunta que reducirá el conjunto de todos los coches que hay en el mercado a unos cuantos que cumplen unos requisitos básicos. Posteriormente con este conjunto más reducido se trabajará con otras consideraciones.

Concepto de filtro

Este procedimiento se denomina *filtro* y es muy frecuente en aplica-

ciones informáticas y en tareas de gestión. Por ejemplo, los alumnos que residen en Barcelona, los clientes que me deben más de 10000 unidades monetarias.

Se denomina filtro, porque filtra la información disponible en una información que me interesa en un momento determinado y descarta la que no me interesa, porque no cumple la condición establecida.

Volviendo al hilo de la cuestión, también se da la distributividad del OR respecto del AND, es decir,

A AND (B OR C) ≡ (A AND B) OR (A AND C)

También aquí el resultado de hacer un AND entre un operando (A) y el resultado de una operación OR entre dos operandos (B y C) es idéntico a hacer primero el AND entre A y B, el AND entre A y C y de éstos dos resultados hacer el OR.

Continuando con el ejemplo anterior, la pregunta que formulamos ahora es:

¿El motor es Diesel y (lleva frenos de disco) o (lleva tracción trasera)?

En esta primera forma de plantearla, es crucial que el motor sea Diesel y además debe llevar una de las dos proposiciones: llevar frenos de disco o llevar tracción trasera, o ambas a la vez.

La otra forma de hacer la misma pregunta que corresponde a la segunda parte de la identidad, es:

¿(El motor es Diesel y lleva frenos de disco) o (El motor es Diesel y lleva tracción trasera)?

Para que el resultado sea verdadero debe ser el m otor Diesel, pues así cada una de las subpreguntas podrá ser cierta. Además para que al menos una de las subpreguntas sea cierta es necesario que lleve frenos de disco y entonces es cierta la primera subpregunta. O que lleve tracción trasera y es cierta la segunda subpregunta. Evidentemente si tiene las dos propiedades, son ciertas las dos subpreguntas. Como estas subpreguntas están conectadas por un *o*, entonces con que sea cierta una de ellas es suficiente.

La igualdad en las tablas de verdad nos prueba la equivalencia

La comprobación de estas propiedades se puede hacer verificando que las tablas de verdad son idénticas. Para ello es más sencillo construir un programa que calcule el valor de las cuatro expresiones

A OR (B AND C)

(A OR B) AND (A OR C)

A AND (B OR C)

(A AND B) OR (A AND C)

entrando los valores de las tres variables booleanas.

Realizando el cálculo para los ocho casos posibles calculará la tabla de verdad de cada expresión, comprobará a continuación que los resulta-

dos de la primera y segunda expresión coinciden entre sí, y que los de la tercera y cuarta expresión también coinciden entre sí.

Intente realizar un programa que se ajuste al formato de impresión siguiente:

```
         Valores de A,B,C :? ? ?
         A AND ( B OR C) :?
 (A AND B) OR (A AND C) :?
          A OR ( B AND C) :?
  (A OR B) AND (A OR C) :?
```

en donde los interrogantes significan los valores a escribir.

La versión que le ofrecemos es:

Tabla de verdad para la distributividad

```
NEW
10 LET V$="Valores A,B,C:"
20 LET E$="A OR ( B AND C):"
30 LET F$="(A OR  B) AND (A OR C):"
40 LET G$="A AND (B OR C):"
50 LET H$="(A AND B) OR (A AND C):"
60 IF LEN(V$)<27 THEN LET V$=" "+V$:GOTO 60
70 IF LEN(E$)<27 THEN LET E$=" "+E$:GOTO 70
80 IF LEN(F$)<27 THEN LET F$=" "+F$:GOTO 80
90 IF LEN(G$)<27 THEN LET G$=" "+G$:GOTO 90
100 IF LEN(H$)<27 THEN LET H$=" "+H$:GOTO 100
200 INPUT "A:",A$
210 IF A$<>"S" AND A$<>"N" THEN GOTO 200
220 INPUT "B:",B$
230 IF B$<>"S" AND B$<>"N" THEN GOTO 220
240 INPUT "C:",C$
250 IF C$<>"S" AND C$<>"N" THEN GOTO 240
260 LET A=A$="S":LET B=B$="S": LET C=C$="S"
300 CLS
310 PRINT V$;A;" ";B;" ";C
320 PRINT E$;A OR (B AND C)
330 PRINT F$;(A OR B) AND (A OR C)
340 PRINT G$;A AND ( B OR C)
350 PRINT H$;(A AND  B) OR (A AND C)
400 GOTO 200
```

Las instrucciones de la 10 a la 100 sirven para preparar los rótulos según el formato de impresión que se ha impuesto. De la instrucción 10 a la 50 se asignan simplemente los valores de los rótulos a las variables textuales V$, E$, F$, G$. De las instrucciones 60 hasta la 100 se añaden tan-

tos blancos a la derecha cuantos son necesarios para completar una longitud de 27. La técnica consiste en ir repitiendo la instrucción de añadir un blanco a la derecha hasta que se alcanza la longitud deseada.

De la instrucción 200 hasta la 250 se realiza la entrada de datos en forma de una variable textual que contiene una S o una N, en caso contrario la entrada es rechazada y se nos vuelve a pedir.

La instrucción 260 transforma las variables textuales en variables lógicas.

El bloque de instrucciones desde la 300 hasta la 350 imprime los resultados y la instrucción 400 recicla el programa a la entrada.

Al ejecutar el programa debe comprobar la propiedad de distributividad, observando que la primera expresión y la segunda coinciden entre sí y la tercera y la cuarta también coinciden entre sí, sea cual sea la combinación de entrada que Ud. coloque.

Para hacer la comprobación exhaustiva debe entrar sistemáticamente todas las combinaciones posibles, si no recuerda como hacerlo consulte las tres primeras columnas de la figura 14.

6.5.3 Leyes de Morgan

El enunciado formal de estas propiedades son:

NOT (A AND B) ≡ (NOT A) OR (NOT B)

y

NOT (A OR B) ≡ (NOT A) AND (NOT B)

Antes de pasar a considerar ejemplos fíjese que la negación, es decir, la aplicación de un NOT, a una operación AND u OR, consiste en negar cada uno de los operandos, aparecen NOT A y NOT B, y cambiar el AND por el OR y viceversa.

Consideremos los ejemplos que se han visto al inicio de la lección sobre el AND y el OR.

En el caso del AND para salir de excursión es necesario que haga sol *y* tenga el coche reparado.

Negar esta condición, es decir, para *no* salir de excursión es necesario que *no* haga sol *o no* tenga el coche reparado.

Hemos aplicado la ley de Morgan, para negar la pregunta global de ir de excursión; hemos negado cada uno de los componentes, no haga sol y no tenga el coche arreglado, y hemos sustituido el *o* por el *y*.

En el caso del OR, la pregunta que formulábamos era para ir sin mojarme por la calle, si queremos negar esta condición, es decir, ir mojándome por la calle debemos negar tomo el paraguas o tomo el impermeable. Es decir, debo responder a

no (tomo el paraguas *o* el impermeable)

en lenguaje natural esta negación se transformaría en,

no tomo el paraguas *y no* tomo el impermeable.

Como negar preguntas que son expresiones

Hemos aplicado otra vez la ley de Morgan. Efectivamente, hemos negado «tomo el paraguas»; decimos no tomo el paraguas, y hemos negado «tomo el impermeable»; es decir, no tomo el impermeable. Finalmente hemos sustituido la unión *o* por la conjunción *y*.

Con un programa similar al caso de la distributividad, puede comprobar exhaustivamente, que las tablas de verdad coinciden al aplicar las leyes de Morgan. Inténtelo Ud. antes de ver nuestra versión. En este caso el número de cálculos a hacer es más corto pues sólo hay dos variables, y, por lo tanto, sólo cuatro posibilidades.

Tablas de Verdad para las leyes de Morgan

```
NEW
 10 LET V$="Valores A,B:"
 20 LET E$="NOT (A AND B):"
 30 LET F$="(NOT A) OR (NOT B):"
 40 LET G$="NOT (A OR B):"
 50 LET H$="(NOT A) AND (NOT B):"
 60 IF LEN(V$)<27 THEN LET V$=" "+V$:GOTO 60
 70 IF LEN(E$)<27 THEN LET E$=" "+E$:GOTO 70
 80 IF LEN(F$)<27 THEN LET F$=" "+F$:GOTO 80
 90 IF LEN(G$)<27 THEN LET G$=" "+G$:GOTO 90
100 IF LEN(H$)<27 THEN LET H$=" "+H$:GOTO 100
200 INPUT "A:",A$
210 IF A$<>"S" AND A$<>"N" THEN GOTO 200
220 INPUT "B:",B$
230 IF B$<>"S" AND B$<>"N" THEN GOTO 220
240 LET A=A$="S": LET B=B$="S"
300 CLS
310 PRINT V$;A;" ";B
320 PRINT E$;NOT (A AND B)
330 PRINT F$;(NOT A) OR (NOT B)
340 PRINT G$;NOT ( A OR B)
350 PRINT H$;(NOT A) AND (NOT B)
400 GOTO 200
```

Debe observar que los resultados de la segunda y tercera línea de la pantalla son idénticos entre sí y también los de la cuarta y quinta sea cual sea la combinación de los datos de entrada.

Esta ley tiene más aplicación en informática de lo que pueda parecer a primera vista. Quizá de todas las propiedades mencionadas es la que más se utiliza en la práctica. Veamos dos ejemplos clásicos.

El primero se refiere al caso de alternativas válidas, por ejemplo, si la entrada es una S o una N, como se ha utilizado en los programas anteriores (vea la instrucción 210 del último programa). La pregunta se formula como

```
210 IF A$<>"S" AND A$<>"N" THEN GOTO 200
```

Es frecuente que por razones de modificación de un programa se deba negar la pregunta, para que la parte del THEN haga otra cosa. Se puede utilizar entonces la pregunta del modo siguiente:

```
210 IF NOT ( A$<>"S" AND A$<>"N") THEN....
```

(detrás del THEN ser haría otra cosa de lo que hace actualmente el programa).

Evidentemente la construcción es correcta tal como está, pero si aplicamos la ley de Morgan del AND, la pregunta anterior es equivalente a

```
NOT(A$<>"S") OR NOT(A$<>"N")
```

si aplicamos ahora el hecho de que los operadores relacionales se cambian al contrario cuando se niegan se obtiene

```
A$="S" OR A$="N"
```

(recuerde que el contrario de la desigualdad es la igualdad).

El segundo ejemplo, se refiere al caso de intervalos numéricos. En programación son frecuentes las preguntas de si un número ESTA FUERA de un intervalo. Por ejemplo, en el primer programa del apartado 6.5.1, se formula la pregunta A<26 OR 54<A. En aquel caso se resolvía mediante dos instrucciones IF, pero se puede formular con la instrucción siguiente:

```
IF A<26 OR 54<A THEN GOTO 200
```

la respuesta a esta pregunta es afirmativa cuando el número está fuera del intervalo.

Si por cualquier razón la queremos negar, podemos colocar un NOT, de modo que la instrucción queda como

```
IF NOT(A<26 OR 54<A) THEN .....
```

Si ahora aplicamos la ley de Morgan esta pregunta se transforma en

```
IF NOT (A<26) AND NOT (54<A) THEN ....
```

si a continuación se aplica la negación de los operadores relacionales se obtiene

```
IF A>=26 AND 54>=A THEN .....
```

o en su forma equivalente

```
IF 26<=A AND A<=54 THEN .....
```

que es la manera típica de plantear la pregunta de si un número ESTA en un intervalo.

Estos cambios de preguntas de afirmativas a negativas son frecuentes en la realización de programas. Por lo tanto, las leyes de Morgan nos ayudan a pasar de una forma de expresión a su contraria.



6.6 LOS OPERADORES LÓGICOS SOBRE NÚMEROS

Hemos dicho al inicio de este capítulo que las variables lógicas las debe controlar el programador pues no se diferencian en nada de las variables numéricas desde el punto de vista de escritura. Sí se diferencian las numéricas y las textuales entre sí, pues estas últimas finalizan con el símbolo dólar ($).

El programador debe controlar el contenido de las variables lógicas en cuanto a su significado, de modo que sean números que sólo contengan un SI o un NO, para que realmente su significado sea lógico.

Sin embargo, se pueden cometer errores y de hecho se cometen frecuentemente, y se aplican operadores lógicos sobre variables numéricas, que contienen un valor distinto de SI y NO como la lógica requiere.

Vamos a describir qué es lo que ocurre cuando se aplican los operadores lógicos sobre números cualesquiera. Para tomar alguna referencia trabajaremos con los números 25 y 30.

Para poder explicar cómo actúan estos operadores, es necesario recurrir a la representación binaria de los números. En general, estos operadores actúan sobre la representación binaria realizando la operación lógica entre pares de bits que comparten el mismo orden. Consideran que el bit 0 es un NO y el bit 1 es un SI.

Tomemos el AND y calculemos 25 AND 30. Lo primero que debemos

hacer es pasar éstos a su representación binaria, evidentemente esto no lo hace el ordenador porque ya tiene todos los números en binario, así

	54321 N.° de orden de los bits
25 =	11001
30 =	11110

Los operadores lógicos actúan sobre los bits

(Si no recuerda cómo se hace, repase el capítulo 3, en donde, hay el método de obtener las representaciones binarias).

Como puede ver el número de orden va de izquierda a derecha. Así para el número de orden 1 en el 25 escrito en binario corresponde un 1 y en el 30 corresponde un cero.

Pues bien el operador AND toma primero el primer bit y realiza la operación del AND entre ellos, siempre el 1 es el SI y el 0 el NO. En nuestro caso, el primer bit de 25 es 1 y el primer bit de 30 es 0. Luego, el AND entre un SI y un NO es un NO; por lo tanto, el resultado de esta primera operación es un 0.

Luego toma el segundo bit, en nuestro caso es un 0 y un 1 y realiza el AND, el resultado es 0.

Sucesivamente va recorriendo cada uno de los bits y realizando la operación. En el caso de 25 AND 30 el resultado es el 11000 en binario, que traducido a decimal sería 24. Por tanto, el resultado de aplicar el AND a estos dos números sería:

24 = 11000

La cadena de bits obtenida de esta manera se transforma en número en representación decimal para entenderlo nosotros, el ordenador lo necesita en binario, y se obtiene el 24 (también en el capítulo 3 encontrará los detalles para transformar el binario en decimal.

El orden de los bits va de derecha a izquierda

Una advertencia; como observará y ya le indicamos más arriba se habla de orden de bits. El primer bit es el que está más a la derecha, contrariamente a lo que puede pensar. El orden es el mismo que las cifras en decimal, primero se consideran las unidades, luego las decenas, las centenas, millares, etc.

Por lo tanto, la operación AND entre dos números cualesquiera consiste en hacer el AND entre los bits de la representación binaria de estos números, tomando como NO el bit que se representa por 0 y como SI el bit que se representa como 1.

Lo mismo que se ha dicho del AND se puede decir del OR. El OR entre dos números cualesquiera consiste en hacer sucesivamente el OR de cada uno de los bits de su representación binaria. En el ejemplo anterior, 25 OR 30 da

	54321 N.° de orden de los bits
25 =	11001
30 =	11110
	______OR
31 =	11111

Como puede observar el resultado son todo 1, pues no hay ningún par de bits que sean ambos cero. Esto en representación decimal es el número 31.

También todo lo dicho es aplicable al XOR. Si se hace el ejemplo de 25 XOR 30 se obtiene

```
          54321 N.° de orden de los bits
25 =      11001
30 =      11110
          _____XOR
 7 =      00111
```

Se observa que aparecen ceros cuando los dos bits son iguales y aparece un 1 cuando los dos bits son distintos. La representación decimal del resultado es 7.

El caso del NOT tiene una lógica sencilla pero su aplicación se complica por una característica de la representación de los números en el ordenador.

En primer lugar, la aplicación del NOT consiste en cambiar todos los bits del número, es decir, los que son 1 pasan a 0 y los que son 0 pasan a 1.

Siguiendo con el ejemplo el NOT de 25 sería

```
          54321 N.° de orden de los bits
25 =      11001
          _____NOT
6 =       00110
```

En principio, la manera parece sencilla, pero en realidad nos hemos olvidado de algunos ceros. Normalmente los números en el ordenador ocupan 16 bits, por lo tanto, los bits 6, 7, 8... 16 son cero y al aplicarles el NOT hay que cambiarles. Da igual que el número de bits sea distinto de 16 el efecto es el mismo que el que se explica aquí.

Por lo tanto, la operación correcta sería

```
          1111 111
          6543 2109 8765 4321 N.° de orden de los bits
25 =      0000 0000 0001 1001
          ___________________            NOT
65510 =   1111 1111 1110 0110
```

Los números unos que están sobre las cifras 6, 5, 4, etc. son sólo para indicar que son el 16, el 15, el 14, etc. Esto mismo se lo encontrará en la figura 17.

La representación decimal de esta tira de bits es 65510, en efecto es el resultado de hacer la operación

```
1*2^15+1*2^14+1*2^13+1*2^12+1*2^11+1*2^10+1*2^9
+1*2^8+1*2^7+1*2^6+1*2^5+0*2^4+0*2^3+1*2^2+1*2+0
```

un poco larga, pero que da como resultado el 65510.

Figura 17 (a) Suma binaria entre 1 y −1 con representación correcta. (b) Suma binaria entre 1 y −1 con representación incorrecta. (c) Suma binaria entre −30 y 25

	1	1111 111	
Bit n.º	7	6543 2109 8765 4321	
Arrastres	1	1111 1111 1111 111	
Número 1		0000 0000 0000 0001	
Número −1		1111 1111 1111 1111	+
Resultado	1	0000 0000 0000 0000	

(a)

	1	1111 111	
Bit n.º	7	6543 2109 8765 4321	
Arrastres		1	
Número 1		0000 0000 0000 0001	
Número −1		1000 0000 0000 0001	+
Resultado		1000 0000 0000 0010	

(b)

	1	1111 111	
Bit n.º	7	6543 2109 8765 4321	
Arrastres			
Número −30		1111 1111 1110 0010	
Número 25		0000 0000 0001 1001	+
Resultado		1111 1111 1111 1011	(Número negativo)
Cambio bits		0000 0000 0000 0100	
Sumo 1		0000 0000 0000 0101	(Número 5)

El resultado es −5.

(c)

Se ha de notar que en el AND, el OR y el XOR este efecto no aparece, pues como los dos números tienen ceros hasta el bit 16, el AND, el OR y el XOR de dos ceros es cero, por lo tanto, no nos importa para nada los ceros que arrastren ambos números.

Los números negativos en el ordenador

Volviendo al NOT, debemos decir, que el resultado obtenido anteriormente no es correcto debido a una complicación adicional que utiliza el ordenador para representar los números negativos.

En efecto, el ordenador utiliza el último bit, en nuestro caso el 16, para saber si un número es positivo o negativo. Cuando este bit está a cero el número es positivo mientras que cuando está a 1 es negativo. Simplemente es necesario rehacer el cálculo anterior de la forma siguiente:

$$-(1*2^{14}+1*2^{13}+1*2^{12}+1*2^{11}+1*2^{10}+1*2^{9}$$
$$+1*2^{8}+1*2^{7}+1*2^{6}+1*2^{5}+0*2^{4}+0*2^{3}+1*2^{2}+1*2+0)$$

en donde se ha sustituido el $1*2^{15}$ por un −, pues éste es el significado del bit 16, recalculada esta expresión resulta −32742.

Sin embargo, este resultado es incorrecto pues la representación de los números negativos no es tan sencilla como hemos expuesto hasta ahora.

La técnica que se utiliza en ordenadores para realizar el cambio de signo, que condiciona la representación de los números negativos, es la llamada del *complemento a dos.* Veamos como es la mecánica para obtener el complemento a dos. Si considera el número 30, cuya representación binaria con 16 bits es

0000 0000 0001 1110

El complemento a dos es el cambio de signo

el complemento a dos se realiza en dos pasos, en el paso 1 se cambian todos los bits, se obtiene en nuestro ejemplo

Paso 1: 1111 1111 1110 0001

en el segundo paso se añade 1, en nuestro ejemplo,

Paso 2: 1111 1111 1110 0010

observe que como estamos sumando en binario, el primer bit es 1 más uno que le sumamos da 10, coloco un cero y llevo una, que sumo al segundo bit que es cero y el resultado es 1.

En el ordenador el complemento a dos de un número significa *cambiarle de signo.*

Así para 1, cuya representación binaria es

(1): 0000 0000 0000 0001

cambiarlo de signo significa hacer el complemento a dos, es decir,

1111 1111 1111 1110: Paso 1
(−1): 1111 1111 1111 1111: Paso 2

y no como cabría esperar del hecho de que el bit 16 signifique el signo, la representación

1000 0000 0000 0001

Por lo tanto, la representación

1111 1111 1111 1111

corresponde al número −1.

Del mismo modo, cambiar de signo el número 30, cuya representación binaria es

0000 0000 0001 1110

consiste en hacer el complemento a dos del modo siguiente:

1111 1111 1110 0001 Paso 1

1111 1111 1110 0010 Paso 2

que será la representación del número −30.

La representación de −30 con el bit 16 a 1 y el resto de bits igual que las del número positivo es incorrecta:

1000 0000 0001 1110

La operación de cambio de signo es simétrica, es decir, si volvemos a cambiar de signo el resultado es el inicial, por ejemplo, si cambiamos de signo a −30 obtenemos 30, en efecto.

1111 1111 1110 0010	(−30)
0000 0000 0001 1101	Paso 1
0000 0000 0001 1110	Paso 2

recuerde que 1+1 en binario se escribe 10; es decir, 1+1 escribiremos cero y llevamos 1.

También si cambiamos el signo de −1, el resultado es 1

1111 1111 1111 1111	(−1)
0000 0000 0000 0000	Paso 1
0000 0000 0000 0001	Paso 2

Por lo tanto, dos veces el complemento a dos deja el número sin cambiar, por esto se puede utilizar este sistema para cambiar de signo.

Es necesario aprovechar el sumador binario para los número negativos

La razón es que de esta manera se puede aprovechar el mecanismo de suma binaria del ordenador para números positivos también para sumar números negativos.

En efecto consideremos el caso de que queremos sumar 1 y −1, si se utiliza el complemento a dos la forma de sumar es la misma que si el número no tuviera signo. La parte (a) de la figura 17 muestra la mecánica de esta suma.

Los dos primeros bits suman 10 (1+1), con lo que llevamos una. En el segundo hay un 0+1, que da uno más la que llevamos de 10, con lo que debemos colocar un cero y llevar una. Esta que llevamos se propaga a lo largo de todos los bits hasta que llegamos al 16, en este mommento nos quedamos con la que llevamos, pero como el ordenador sólo tiene 16 bits la perdemos y el uno que aparece en la posición 17 desaparece.

Si imaginamos una representación distinta, por ejemplo, la que cambia sólo el bit 16 para indicar los números negativos, no se puede aplicar este sumador que tiene el ordenador, sino que se precisaría otro tipo de sumador para los números negativos. Observe que en la parte (b) de la figura 17, está realizada la suma si la representación no es el complemento a dos. Como puede ver el resultado sería −2, dentro de la misma representación, claro.

Se prefiere, por lo tanto, complicar la representación de los números con signo que tener que diseñar un sumador distinto para números positivos y para números negativos.

En la parte (c) de la figura 17, encontrará el ejemplo de sumar −30 y 25, observe que con una simple suma hemos conseguido el resultado de −5.

Volviendo a la cuestión, se ha introducido todo el problema de cambio de signo, porque deseábamos saber cual es el número que da la aplicación del NOT al número 25, que era

1111 1111 1110 0110

Como tiene el bit 16 en uno, sabemos que el número es negativo pero no sabemos cuanto vale, para ello lo cambiaremos de signo y nos dará un número positivo y sabremos cual es su parte positiva, en efecto.

1111 1111 1110 0110	NOT 25
0000 0000 0001 1001	Paso 1
0000 0000 0001 1010	Paso 2

el número que hemos obtenido al cambiar de signo es el 26, luego el NOT de 25 es −26.

Cómo calcular el NOT en la aritmética decimal

Si analizamos el procedimiento de cambio de signo, o lo que es lo mismo el complemento a dos, lo primero que se hace es cambiar los bits, precisamente la operación de NOT, y luego se añade 1. Expresado en forma de ecuación matemática

```
-A = NOT A + 1
```

en donde, A es un número cualquiera. Leído en lenguaje natural diremos que cambiar de signo es el NOT del número más la unidad.

Si se manipula algebráicamente, es decir, el 1 se traslada al otro lado, obtenemos la ecuación

```
NOT A = -A - 1.
```

Esta fórmula permite calcular directamente el NOT de 25 que será −25 −1 = −26.

La figura 18 nos muestra dos ejemplos de aplicación de la fórmula para los números 5 y −3.

En resumen y como norma general, las operaciones lógicas sobre variables numéricas se realiza sobre cada uno de los bits de su representación binaria, tomando el 0 como NO y el 1 como SI.

Figura 18 Detalles de cálculo del NOT de 5 y −3

NOT 5 = −5 −1 = −6 Fórmula

	Número 5	0000 0000 0000 0101
	NOT de 5	1111 1111 1111 1010
		Resultado negativo Hay que cambiar de signo
Cambio bits		0000 0000 0000 0101
Sumar 1		0000 0000 0000 0110
		Número 6

NOT −3 = −(−3)−1 = 2

		¿Cómo se representa El número −3?
	Número 3	0000 0000 0000 0011
Cambio bits		1111 1111 1111 1100
Sumo 1		1111 1111 1111 1101
		El número −3
	Número −3	1111 1111 1111 1101
	NOT −3	0000 0000 0000 0010
		Número positivo Resultado 2

Otra cuestión relacionada de la que ya hemos hablado en el apartado 6.1.2 es la de la forma del IF tal como

```
60 IF A THEN PRINT "SI"
```

que aparece en el programa que pregunta si una persona es mujer rubia, y en donde, A es una variable lógica.

El IF se realiza si el número que viene a continuación es distinto de cero

Es el momento de señalar qué ocurre si esta variable, que debería ser lógica, es por error numérica. Sólo se ejecuta la parte del THEN siempre que la variable sea distinta de cero. Es decir, sólo se salta el PRINT si la variable A es nula, este comportamiento está de acuerdo con la definición de variables lógicas.

El peligro más frecuente ocurre cuando se plantea del modo siguiente la pregunta

```
IF NOT A THEN PRINT "NO"
```

pues sólo se salta la parte del THEN si A vale −1.

Este ejemplo nos lleva a otra cuestión de la que hemos hablado varias

Figura 19 Tabla del AND y del OR numérico, según la representación del SI y el NO en las variables lógicas

Representación	
NO = 0 SI = −1	NO = 0 SI = 1
−1 AND −1 = −1 −1 AND 0 = 0 0 AND −1 = 0 0 AND 0 = 0	1 AND 1 = 1 1 AND 0 = 0 0 AND 1 = 0 0 AND 0 = 0
−1 OR −1 = −1 −1 OR 0 = −1 0 OR −1 = −1 0 OR 0 = 0	1 OR 1 = 1 1 OR 0 = 1 0 OR 1 = 1 0 OR 0 = 0

veces, ¿Por qué la representación del SI se hace en algunos ordenadores con 1 y en otros con −1?

La contestación es que la elección de −1 permite que el NOT −1 sea directamente NO, en cambio la elección de 1 no lo permite pues ya sabemos que el NOT 1 es −2. Por lo tanto, en los ordenadores que utilizan el 1 como representación del SI, el NOT no puede comportarse como hemos explicado aquí.

Por otra parte, la elección del 1 está más acorde con los símbolos utilizados por los teóricos de la lógica matemática, asociada con el álgebra de Boole.

Por lo que respecta a los demás ordenadores da igual elegir el SI como 1 o −1, son ambos coherentes entre sí. La figura 19 da la tabla de AND y OR cuando se escoge la representación 1 o −1, observe que en la columna en que se utiliza el cero y el −1, sólo aparece el 0 y el −1 y lo mismo para la columna del 1.

RESUMEN

Los operadores lógicos son importantes para concretar preguntas de tomas de decisión realizadas en el lenguaje natural, al lenguaje propio del ordenador. El mecanismo más importante para pasar las preguntas del lenguaje natural al del ordenador consiste; en reducir nuestras preguntas en lenguaje ordinario a preguntas binarias y luego encadenarlas mediante la unión, partícula *o*, y la conjunción, la partícula *y*. Utilizando además la negación como recurso adicional.

Los casos más frecuentes de utilización en programación de los opereadores lógicos son el tratamiento de alternativas múltiples; básicamente el mecanismo consiste en encadenar las distintas alternativas mediante la unión.

Si se aplica, por ejemplo, a considerar unos determinados colores la pregunta se formula como

¿El traje es amarillo *o* el traje es azul *o* el traje es negro?

Y si se aplica al tratamiento de intervalos numéricos (básicamente cuando hay que saber si un número está comprendido en un intervalo de valores, digamos límite superior y límite inferior), la pregunta se plantea como

¿El límite inferior es menor o igual que el número *y* el número es menor o igual que el límite superior?

Con estas dos formulaciones básicas se pueden resolver problemas más complicados en que intervengan un número de variables más elevado.

Cada operador relacional tiene su contrario, al aplicar la negación sobre una pregunta formulada con un operador relacional. Así, se puede sustituir la negación por el operador relacional contrario.

Con los contrarios de los operadores relacionales y sabiendo cómo se pueden colocar las preguntas en un IF, se puede evitar el uso de los operadores lógicos. Por otra parte, esto es poco recomendable ya que dan mayor claridad al texto de programa.

Una de las propiedades más importantes de los operadores lógicos es la distributividad del AND respecto al OR, que significa que si hacemos un AND en donde una de las preguntas es a su vez un OR, se puede sustituir por dos subpreguntas con un OR, conjuntadas globalmente con un AND.

También se da la distributividad del OR respecto al AND; es decir, que si hacemos un OR y una de las preguntas es a su vez un AND, se puede sustituir por dos subpreguntas con un AND, unidas globalmente con un OR.

Las leyes de Morgan permiten la negación de una pregunta que se compone de dos subpreguntas unidas por un operador AND u OR. El resultado es negar cada una de las subpreguntas y cambiar el AND por un OR o cambiar el OR por el AND.

Muchas aplicaciones en informática utilizan preguntas para seleccionar parte de la información disponible. Este tipo de programas se denominan filtros, ya que su misión es reducir el conjunto de informaciones a unas que cumplan una determinada condición.

En principio, los operadores lógicos actúan sobre variables lógicas. Sin embargo, como las variables lógicas son variables numéricas es posible que por error aparezcan números cualesquiera en los operadores lógicos.

Cuando así sucede, lo que ocurre es que se realizan las operaciones lógicas sobre cada uno de los bits de los números en cuestión en representación binaria.

Para hacer estas operaciones entre los bits se toma el 0 como el NO y el 1 como el SI.

Para poder preveer el resultado, es necesario conocer profundamente la transformación de representación decimal de un número a representación binaria.

El hecho de que el ordenador tiene un sumador para números positivos en representación binaria, obliga a representar los números negativos de tal manera que se pueda aprovechar este sumador para realizar las operaciones típicas de números negativos.

Se utiliza, para ello, la técnica del complemento a dos para el

cambio de signo, que permite, gracias a que el ordenador tiene un número fijo de bits, utilizar el sumador de números positivos para operar con números negativos.

Esta propiedad del complemento a dos permite establecer la fórmula para calcular el NOT de un número como un cambio de signo menos una unidad.

También el caso particular del NOT hace que la representación del SI en muchos ordenadores sea precisamente el −1 pues es el NOT de 0, y el NOT de 0 es −1.

Finalmente, cuando en un IF se coloca un número en lugar de una expresión lógica, sólo se realiza la parte del THEN si el número es distinto de cero.

EJERCICIOS DE AUTOCOMPROBACION

Complete las frases siguientes:

27. El operador NOT se traduce al lenguaje ordinario como ....
....

28. Para realizar decisiones con el ordenador es necesario reducir las preguntas a formulaciones ....

29. El operador .... se traduce al lenguaje ordinario como *o.*

30. Cuando hay que decidir sobre si compro una libreta y compro un lápiz utilizo el operador ....

31. Cada operador relacional tiene su ....

32. Dos propiedades de los operadores booleanos son la ....
.... del AND respecto al OR y viceversa y las leyes de Morgan.

33. Un .... es un tipo de toma de decisión para seleccionar información de un conjunto más amplio.

34. Las leyes de Morgan sirven para el cálculo de las ....
.... de las expresiones con AND u OR.

35. Las operaciones lógicas con variables numéricas consisten en hacer operaciones lógicas entre ....

36. Para el cambio de signo en los ordenadores se utiliza la técnica del ....

Encierre en un círculo la letra que corresponde a la alternativa correcta.

37. La negación de la expresión NOT (A 5 OR A 7) es:

a) NOT A < 5 AND A > 7

b) NOT A < 5 OR NOT A > 7

c) A < 5 AND A <= 7

d) A >= 5 AND A <= 7

38. ¿Qué expresión es equivalente a: (A AND B) OR C?

a) A AND C OR A OR B

b) (A OR C) AND (B OR C)

c) A OR C AND B OR C

d) A AND (B OR C)

39. La técnica de complemento a dos para cambiar el signo se utiliza para:

a) Ir más rápido.

b) Aprovechar el sumador de números positivos.

c) Evitar números muy grandes.

d) Hacer el NOT.

40. Las operaciones lógicas entre variables numéricas consisten en operar sobre:

a) Los bits de la representación binaria.

b) Los valores absolutos.

c) No se puede operar, pues da error la máquina.

d) Es como una multiplicación.

41. Detrás del IF puede ir una variable numérica. El programa realiza la parte del THEN cuando:

    a) El valor es cero.

    b) Nunca.

    c) Si es positivo menor que 100.

    d) Si es distinto de cero.

42. Realizar la tabla de verdad de la expresión

    (A AND NOT B) OR (NOT A AND B)

    y compararla con la del XOR.

43. Realizar la tabla de verdad de la expresión

    NOT (NOT A OR NOT B)

    y compararla con la del AND.

44. Hallar el complemento a dos del número 15 en binario.

45. Hallar el complemento a dos del número −35 en binario.

Hallar utilizando los números binarios los resultados siguientes:

46. 7 AND 8

47. 2 OR 3

48. 2 OR (5 AND 4)

49. 10 OR 5

50. 5 AND 4



Capítulo 7
• La función INKEY.
• Diseño del programa.

ESQUEMA DE CONTENIDO

Objetivos	
Función INKEY$	Funcionamiento Utilización Visualización temporal Selección opciones Reconocimiento de teclas
Diseño de programas	Niveles de diseño Técnicas de diseño Ordinogramas Símbolos utilizados Representación gráfica de las instrucciones
Casos prácticos	Ecuación de segundo grado Actualización de precios Números pares Interés compuesto
Codificación	
Otros símbolos	
Tablas de decisión	Formato de la tabla de decisión Definiciones Casos prácticos Clasificación de piezas Depuración de las tablas

7.0 OBJETIVOS

En este capítulo estudiaremos dos temas bien diferenciados. En primer lugar aprenderemos a utilizar la función INKEY$ que permite obtener un dato del teclado con sólo pulsar una tecla. Esta función es imprescindible en los programas interactivos; es decir, aquellos en los que se establece un diálogo entre el usuario y el ordenador.

El segundo tema es de gran importancia y se refiere al diseño de programas. En este apartado veremos la necesidad del diseño y estudiaremos las distintas técnicas y los sistemas de representación gráfica de problemas. De estos sistemas aprenderemos los dos más conocidos, los ordinogramas y las tablas de decisión. Un buen diseño previo es imprescindible para que un programa funcione con eficiencia. Por esta razón es muy importante conocer estas técnicas.

Entre los programadores profesionales se dice que cuanto antes tecleemos un programa más tarde lo terminaremos. Esto significa que si pone mucha atención a la parte de diseño, la consecución de un programa final correcto se alcanza más rápidamente.

7.1 FUNCION INKEY$

Por el momento, la única forma que conocemos para que el programa obtenga datos del exterior a través del teclado es el empleo de la instrucción INPUT. En principio, parecería que esta instrucción es suficiente para cubrir todas las necesidades de comunicación del operador con el programa. Sin embargo, tiene dos inconvenientes:

Inconveniente del INPUT

— En primer lugar, hay que pulsar siempre la tecla de fin de línea después de teclear el dato. A veces, nos puede interesar que con sólo pulsar una única tecla se desencadene un determinado proceso.

— El segundo inconveniente (en ciertos casos), es que al llegar a una instrucción INPUT, el programa se detiene y espera indefinidamente hasta que le damos los datos que pide. En algunos casos nos puede convenir que el programa consulte el teclado y, si se ha pulsado una tecla, la tome como respuesta. Si no se ha pulsado ninguna, entonces el programa no debe esperar al operador sino que debe seguir.

El INKEY$ es una función no una instrucción

El BASIC tiene precisamente una función especial diseñada para resolver esta cuestión. Esta función se denomina INKEY$. La palabra INKEY proviene del inglés «Input key» que significa entrar una tecla. Conviene remarcar el hecho de que se trata de una función y no de una instrucción. Por consiguiente hay que utilizarla dentro de otras instrucciones como LET o PRINT. Como se puede apreciar por el símbolo dolar ($), su resultado es de tipo textual. Esta función carece de argumentos y, por tanto, no lleva paréntesis a continuación de su nombre.



7.1.1 Funcionamiento

Cuando se ejecuta esta función, consulta el teclado. Si el operador ha pulsado alguna tecla, el símbolo correspondiente a dicha tecla pasa a ser el resultado de la función, sin esperar a que se pulse la tecla de fin de línea. El resultado será de tipo textual y estará formado por un solo símbolo. Por el contrario, si el operador no ha pulsado tecla alguna, la función no espera y entonces el resultado de la misma será el texto vacío (de longitud cero). El siguiente programa nos servirá para demostrar que esta función no espera.

La función INKEY$ no espera

```
10 LET A$=INKEY$
20 PRINT "FIN"
```

En la línea 10 vemos una forma típica de utilizar la función INKEY$. El resultado se almacena en A$. Al efectuar un RUN, el programa escribe inmediatamente la palabra FIN. Esto ocurre porque no se detiene en lá línea 10. El valor de la variable A$ será el texto vacío. De hecho, tal como está construido el programa, no hay tiempo suficiente para que el operador pueda pulsar una tecla después de teclear RUN y fin de línea. Por tanto, la función INKEY$ en este programa devolverá siempre el texto vacío como resultado.

Tal como hemos construido el programa, vemos que es inviable el empleo de esta función. Si queremos asegurar que el operador pulse una tecla, utilizaremos el siguiente procedimiento:

```
10 LET A$=INKEY$
20 IF A$="" THEN GOTO 10
30 PRINT "TECLA=";A$
```

¿Cómo realizar la espera con la función INKEY$?

Las dos primeras instrucciones son la clave del asunto. Mientras el operador no teclee nada, el programa se queda dando vueltas entre estas dos líneas. Si A$ contiene el texto vacío, representado por dos comillas juntas («») el control pasa de nuevo a la línea 10. Este proceso seguirá indefinidamente hasta que pulsemos una tecla. Entonces A$ contendrá precisamente el carácter (letra o símbolo) asociado a la tecla y por tanto la pregunta de la línea 20 tendrá un valor de falso pasando control a la línea 30. En esta línea escribimos el valor de la variable para confirmar que el proceso ha funcionado correctamente.

La variable que recibe el valor de la tecla (A$ en el ejemplo anterior) sólo puede tener longitud cero o uno. Será cero cuando contenga el texto vacío y 1 cuando contenga un símbolo. Longitudes mayores de 1 no son posibles a causa de la propia naturaleza de su funcionamiento.

La función INKEY$ nos devuelve cualquier tecla aunque no sea imprimible

Es importante señalar que, desde el punto de vista de la función INKEY$, todas las teclas son iguales. Esto incluye a las teclas que realizan operaciones especiales como la tecla de fin de línea, la de retroceso o las de movimiento del cursor. Lógicamente, si se pulsan teclas especiales, la función devuelve carácteres no imprimibles. Ya sabemos que estos carac-

teres son los que tienen un número inferior a 32 en el código ASCII. Aunque no se pueden imprimir, sí que se puede averiguar su código. El siguiente programa nos servirá para ello.

```
10 LET A$=INKEY$
20 IF A$="" THEN GOTO 10
30 PRINT "CODIGO=";ASC(A$)
40 GOTO 10
```

Las dos primeras líneas ya las conocemos. En la línea 30 utilizamos la función ASC para obtener el código ASCII de A$. Si efectuamos un RUN y a continuación pulsamos una A, el programa escribirá un 65. Si después pulsamos una tecla especial, obtendremos su código. Estos códigos suelen variar bastante de una máquina a otra, aunque generalmente la tecla con retroceso suele tener el código 8 y la de fin de línea tiene el 10 ó el 13.

7.1.2 Utilización

La función INKEY$ tiene muchas aplicaciones y es casi imprescindible en los programas de funcionamiento interactivo; es decir, que establecen un diálogo con el usuario. Un caso particular de programa interactivo son los juegos por ordenador, en los cuales se utiliza profusamente esta función. Sin embargo, de momento nos limitaremos a aplicaciones más sencillas, puesto que no hemos adquirido todavía los conocimientos suficientes de BASIC para abordar la realización de un programa de videojuego. Los programas de juegos son mucho más difíciles de lo que parece a primera vista.

7.1.3 Visualización temporal

Visualización de ayudas e instrucciones de funcionamiento

Un caso muy frecuente es que el programa muestre en pantalla un texto que debe desaparecer en cuanto el operador lo haya leído. Generalmente son los programas que visualizan en pantalla las instrucciones de uso. Una vez el operador las ha leído, pulsa una tecla y las instrucciones se borran. Un ejemplo de estos programas es el siguiente, que calcula el área de un triángulo.

```
10 PRINT "INSTRUCCIONES PARA"
20 PRINT "UTILIZAR EL PROGRAMA"
30 PRINT
40 PRINT "1.- ENTRE LA BASE"
50 PRINT "2.- ENTRE LA ALTURA"
60 LET A$=INKEY$
70 IF A$="" THEN GOTO 60
80 CLS
```

```
  90 INPUT B
 100 INPUT A
 110 LET S=B*A/2
 120 PRINT "SUPERFICIE=";S
```

Las líneas comprendidas entre la 10 y la 50 son las que imprimen el mensaje. En este caso, las explicaciones son muy reducidas, pero nada impide que se escriban unos comentarios más extensos.

Las líneas 60 y 70 dejan el programa bloqueado hasta que el usuario lea las instrucciones de la pantalla y dé la orden de seguir, pulsando cualquier tecla. La línea 80 borra la pantalla, puesto que ya no son necesarias las instrucciones. A partir de la línea 90 el programa pide los datos de base y altura y calcula la superficie del triángulo.

7.1.4 Selección opciones

A menudo se diseñan programas que tienen varias opciones. Por ejemplo, el programa anterior podríamos ampliarlo añadiendo la posibilidad de calcular áreas de círculos. Al empezar, el usuario pulsa una tecla determinada y el programa se dirige a la opción escogida. Veamos un ejemplo.

Ejemplo de Menú con el INKEY$

```
  10 PRINT "OPCIONES"
  20 PRINT
  30 PRINT "1.- TRIANGULO"
  40 PRINT "2.- CIRCULO"
  50 PRINT "F.- ACABAR"
  60 LET A$=INKEY$
  70 IF A$="1" THEN GOTO 1000
  80 IF A$="2" THEN GOTO 2000
  90 IF A$="F" THEN GOTO 9000
 100 GOTO 60
1000 INPUT "ENTRE LA BASE:",B
1010 INPUT "ENTRE LA ALTURA:",A
1020 PRINT "AREA=";B*A/2
1030 GOTO 10
2000 INPUT "ENTRE EL RADIO:",R
2010 PRINT "AREA=";3.1416*R*R
2020 GOTO 10
9000 CLS : REM Se finaliza el programa.
9010 END
```

El programa es parecido al anterior. La novedad está en las líneas que van de la 60 a la 100. Ahora el programa no sólo espera a que se pulse una tecla, sino que además quiere que ésta sea un 1 ó un 2 ó una F. Como

se ve en el listado, cualquier otra tecla dirige de nuevo el control a la línea 60.

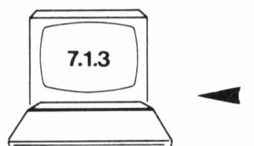
Si se pulsa un 1, el control se dirige a la línea 1000 y el programa evalúa el área de un triángulo, volviendo al inicio en la línea 1030.

Por el contrario, si se pulsa un 2, se dirige a la 2000, donde calcula el área de un círculo de radio R, volviendo después del cálculo al inicio, en la línea 2020.

Finalmente, si se pulsa una F el programa se dirige a la línea 9000, en donde finaliza después de limpiar la pantalla.

Ciertamente, en la línea 60 se hubiera podido emplear una instrucción INPUT, pero entonces el operador estará obligado a usar dos teclas, la de la opción escogida y la tecla de fin de línea.

Este procedimiento de selección de opciones es muy común en informática y recibe el nombre de «Menú». Este nombre proviene de que se presentan una serie de opciones de las cuales se escoge una, al igual que en el menú de un restaurante.



7.1.5 Reconocimiento de teclas

El siguiente programa nos servirá de entrenador para recordar las posiciones de las teclas y así aprender a utilizar el teclado con mayor rapidez. Este programa es interesante puesto que, a pesar de estar construido con pocas instrucciones, su funcionamiento parece dotar al ordenador de una cierta «inteligencia».

```
10 LET L=64
20 LET L=L+1
30 LET T$=CHR$(L)
40 PRINT "PULSE:";T$
50 LET X$=INKEY$
60 IF X$="" THEN GOTO 50
70 IF X$=T$ THEN GOTO 100
80 PRINT "ERROR, HA PULSADO:";X$
90 GOTO 40
100 PRINT "CORRECTO"
110 IF L<90 THEN GOTO 20
```

Este programa nos irá indicando que pulsemos la tecla A, la B, etc. Si no acertamos, nos dará un mensaje de error e insistirá en la misma petición. Sólo debe pulsar la tecla, no es necesario apretar la tecla de fin de línea, si lo hace, el programa dará un error pues no se trata de la tecla que nos ha pedido, de todos modos insistirá en su petición.

El funcionamiento es el siguiente. La variable L empieza en 64 y en la línea 20 se incrementa y vale 65. A continuación (línea 30), este valor se toma como código y se obtiene la letra correspondiente mediante la función CHR$. El valor 65 corresponde a una A mayúscula. Esta letra se almacena en T$. El programa imprime en pantalla el mensaje PULSE: A. Seguida-

mente (líneas 50 y 60) viene el conocido proceso de obtención de una tecla, que se memorizará en X$. En la línea 70, el programa comprueba si hemos tecleado la respuesta correcta. Si no es así, nos informa de la tecla pulsada y pasa control de nuevo a la línea 40 para que lo intentemos de nuevo. Si acertamos, nos escribirá el mensaje CORRECTO. Entonces, puesto que L es inferior a 90, el programa vuelve a la línea 20 donde se incrementa L, valiendo 66. El proceso se repite ahora para la letra B. El programa acabará cuando lleguemos a la Z ya que entonces el IF de la línea 110 no se cumplirá.

Además de para las letras, este programa también sirve para entrenarnos a reconocer los otros símbolos del teclado. Para ello, teniendo presente la tabla de códigos ASCII, modificaremos el valor inicial (el 64 de la línea 10) y el valor final (el 90 de la línea 110) por los valores adecuados.

7.2 DISEÑO DE PROGRAMAS

En los capítulos anteriores ya hemos visto algunos aspectos del diseño de programas. En este capítulo estudiaremos el tema con mayor amplitud.

Elaborar un programa significa, en cierto modo, dar una serie de órdenes para que la máquina realice una tarea. Si bien el ordenador es el que hace el trabajo, está claro que para llevar a cabo esta tarea hay que planear los distintos pasos que deben efectuarlo y esta planificación corresponde al programador. Cuando mejor sea este diseño, mayor será la eficiencia del programa resultante. Incluso, es posible que si el diseño no es correcto, el programa tenga un funcionamiento inesperado en ciertos casos o bien que se produzcan errores.

Por otra parte, cuando se realiza la planificación hay que atender a los aspectos básicos de la cuestión y dejar los detalles para más adelante. Por consiguiente, la fase de diseño se ha de llevar a cabo sin tener en cuenta el lenguaje de programación a utilizar. Como ya sabemos, existen muchos lenguajes para ordenadores. Pero un diseño debe poder trasladarse a cualquiera de ellos. En nuestro caso, el lenguaje será el BASIC, pero las técnicas aprendidas aquí serán utilizables siempre.

Cuando se escribe un programa, utilizando un lenguaje de ordenador, la sintaxis es muy estricta y seguramente habremos comprobado alguna vez que, una simple coma mal situada, puede impedir que el programa funcione correctamente. Si nos dedicamos simultáneamente a tareas de diseño y a la vez a vigilar los detalles sintácticos, probablemente resultará un esfuerzo improductivo. Por esta razón, los programadores decimos que en esta fase hay que dar prioridad a la semántica en lugar de a la sintaxis.

Semántica y Sintaxis

La semántica hace referencia al significado de las cosas. Aquí utilizamos esta palabra refiriéndonos a qué hace cada operación fundamental. Por el contrario, la sintaxis sería la forma en que se escribiría esta operación. Precisamente, por no tener en cuenta la forma, es por lo que decimos que el diseño es independiente del lenguaje a utilizar. Esto no quiere decir que las normas sintácticas haya que olvidarlas, ni mucho menos. Una vez finalizado el diseño habrá que realizar la escritura de las instrucciones reales que compondrán el programa. De hecho, la mayor parte de esta Enciclopedia está dedicada a aprender las normas sintácticas del BASIC.

7.2.1 Niveles de diseño

Diseño jerárquico

Un punto importante a tener en cuenta es el nivel de detalle al que se trabaja. La cantidad de detalles que se tienen en cuenta depende del uso a que se destine. En las primeras etapas del desarrollo de un programa se prueban diferentes enfoques para diseñarlo. En este caso, se agrupan varias operaciones elementales en un solo segmento. Cuando estemos satisfechos del diseño previo, procederemos a ampliar los detalles de cada segmento o bloque. Este proceso lo iremos repitiendo profundizando cada vez más en el nivel de detalle hasta que quede bien claro qué pasos realiza cada operación fundamental y sobre qué datos opera. Este sistema recibe el nombre de diseño jerárquico.

7.2.2 Técnicas de diseño

Todo lo que llevamos dicho hasta ahora, justifica la importancia de la tarea de diseño. Recuerde el refrán en informática que dice que «cuanto antes tecleemos un programa, más tarde lo terminaremos». Esto significa que si empezamos a escribir instrucciones en el teclado, sin habernos parado a reflexionar sobre el problema, es probable que nos pasemos gran cantidad de tiempo haciendo pruebas y arreglos en el programa. Tiempo que seguramente nos habríamos ahorrado con un poco de reflexión previa.

En este texto utilizaremos dos técnicas de diseño: los ordinogramas y las tablas de decisión. Tal vez no sean los mejores, pero sí que son las más sencillas y ciertamente las de uso más amplio, incluso, en programadores profesionales. Por eso es conveniente empezar por ellas.

7.2.3 Ordinogramas

Los ordinogramas son una técnica de diseño basada en el empleo de símbolos gráficos para describir las operaciones que debe realizar un proceso. El uso de gráficos es de gran ayuda para que la comprensión del proceso sea rápida. También tiene la ventaja de que es una forma compacta de describir un programa. Por esta causa, se suelen utilizar los ordinogramas como documentación del programa, de modo que otra persona (o nosotros mismos al cabo de un tiempo), mirando el ordinograma comprendamos enseguida qué operaciones realiza. Este punto es importante, puesto que cuando acabamos de realizar un programa recordamos perfectamente qué operaciones se realizan a cada paso. Sin embargo, si al cabo de un cierto tiempo volvemos a mirar el programa, entonces probablemente habremos olvidado la mayor parte del mismo. Si sólo hemos guardado el listado de las instrucciones, el descifrar el funcionamiento del programa será una tarea ardua y difícil. Esta tarea se vería facilitada en gran medida, si hubiésemos tenido la precaución de guardar un ordinograma que reflejara el funcionamiento del programa, junto con el listado de las instrucciones.

Funciones del ordinograma

En resumen, el ordinograma tiene dos funciones:

a) Como ayuda para el desarrollo y puesta a punto de los programas, sir-

viendo además de guía para la escritura de las instrucciones en un lenguaje de ordenador.

b) Como documentación básica indispensable de un programa.

Los ordinogramas reciben también a veces el nombré de diagramas de flujo (de inglés «Flow chart») puesto que en ellos se describe el sentido de avance (el flujo) de un proceso.

7.2.4 Símbolos utilizados

Ya hemos dicho que los ordinogramas se basan en el empleo de símbolos gráficos. Estos símbolos se muestran en la figura 1. A menudo, a estos símbolos se les denomina bloques.

El flujo del programa

En primer lugar se emplea la *flecha* (Fig. 1.a) para indicar el flujo del proceso. La elección de este símbolo es especialmente acertada, puesto que la flecha tiene un significado muy explícito y que todo el mundo entiende:

Los procesos que realiza el programa

Se emplea un *rectángulo* para representar un proceso en general (Fig. 1.b). Aquí se aplica lo que hemos mencionado antes sobre niveles de diseño. En una primera fase, un rectángulo incluirá globalmente un proceso complicado. Más adelante subdividiremos este bloque en sus procesos parciales, cada uno con su correspondiente rectángulo.

La entrada y salida de datos es un proceso que, debido a su importancia, se le ha asignado un símbolo especial. Este símbolo es un *rectángulo inclinado* (Fig. 1.c) (un paralelepípedo para ser exactos). El ordenador se comunica con el exterior mediante los procesos de entrada-salida. Ya sabemos que, en BASIC, esta comunicación se realiza fundamentalmente mediante las instrucciones INPUT y PRINT.

Cuando se diseña un programa ha de establecerse con claridad dónde empieza el flujo del proceso y dónde acaba. En BASIC, el principio y el fin quedan automáticamente fijados, puesto que las líneas van numeradas en

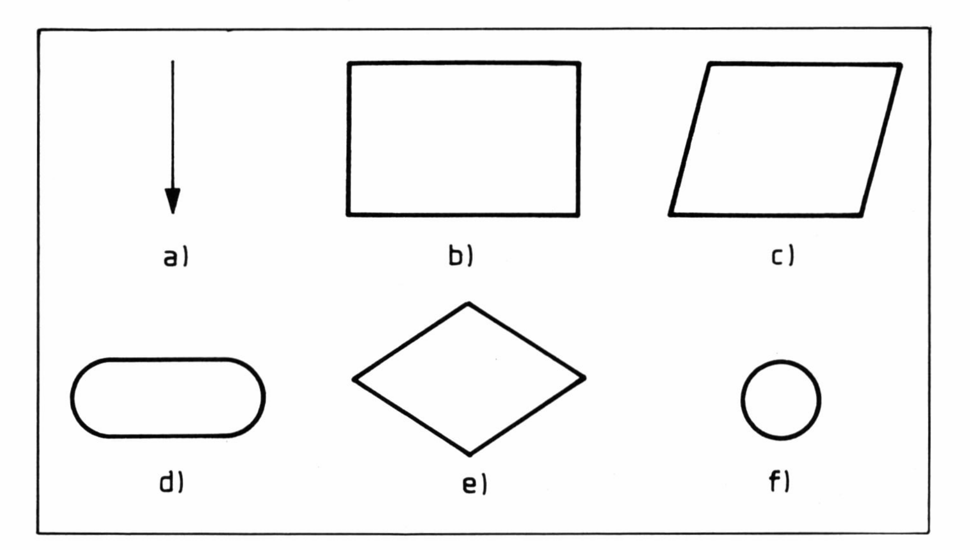


Figura 1: Símbolos empleados en los ordinogramas: a) Flecha. b) Proceso. c) Entrada/Salida. d) Terminal. e) Decisión. f) Conector de página.

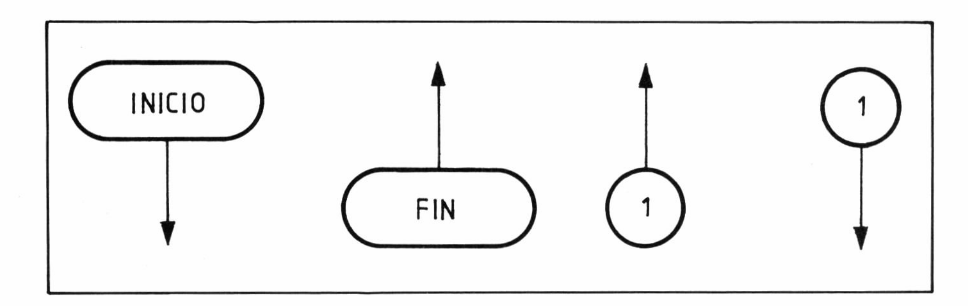


Figura 2: Símbolos que tienen una única conexión.

orden correlativo. Si bien sólo puede existir un único inicio, en cambio puede haber varios finales. En BASIC esto se realiza colocando instrucciones END o STOP en los lugares adecuados. El símbolo empleado en los ordinogramas para representar el inicio o el fin del proceso es el que se muestra en la figura 1.d, es un *rectángulo con los bordes redondeados.*

Se emplea un *rombo* para representar la toma de decisiones (Fig. 1.e). Mediante este símbolo se representan las condiciones que permitirán elegir un camino u otro según la condición sea cierta o falsa. El bloque de decisión equivale a la instrucción IF del BASIC.

Finalmente, se utiliza *el círculo* como conector de página (Fig. 1.f). Ocurre a menudo que un ordinograma no cabe en una sola página y debe continuarse en otra. Entonces, para señalar que una flecha, que termina en una página, enlaza con otra en la siguiente, se utiliza un círculo con un número. Lógicamente este número será el mismo en ambas páginas. Si son varias las flechas que continúan en otra página, se dan números correlativos.

Las conexiones de los símbolos

Un punto importante es cómo se enlazan las flechas con los símbolos. En primer lugar están los símbolos que sólo tienen un punto de unión. Estos símbolos están representados en la figura 2. Son el símbolo terminal (inicio o fin) y el conector de página. La unión con la flecha puede ser de llegada o de salida, pero nunca ambas a la vez.

Otro grupo de símbolos tienen una flecha de entrada y otra de salida. Estos símbolos se muestran en la figura 3 y son el bloque de proceso y el de entrada/salida. El flujo les llega por la flecha de entrada, se efectúa la operación que indican y el flujo continúa por la flecha de salida.

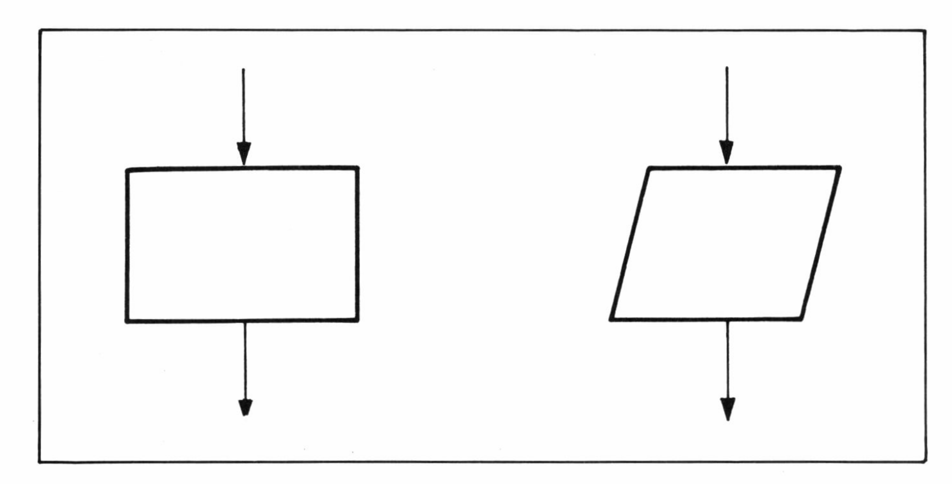


Figura 3: Símbolos de dos conexiones, una de entrada y una de salida.

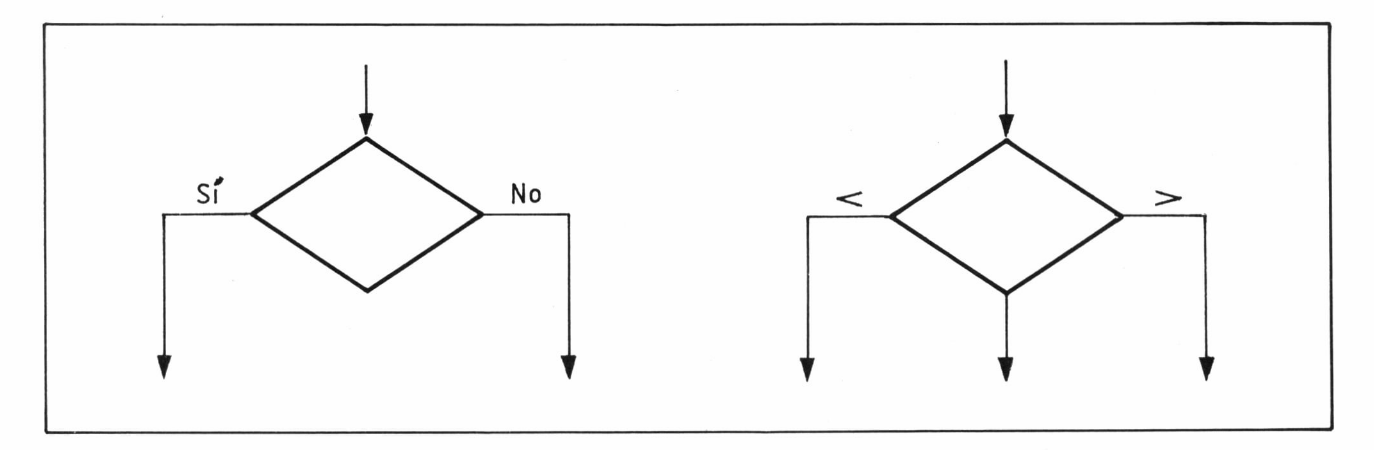


Figura 4: Símbolos con más de una conexión de salida.

El símbolo de decisión tiene una flecha de llegada y, a diferencia de los demás, tiene dos flechas de salida. Por tanto, este símbolo es el único que puede ralizar una bifurcación. El flujo seguirá por una salida o por la otra según el resultado de la condición sea cierto o falso. En algunos casos pueden existir tres salidas que corresponden a una condición con tres resultados posibles: mayor que (>), igual que (=) y menor que (<). En la figura 4 se representa el bloque de decisión con dos y tres salidas.

Un punto importante, que conviene recalcar, es que los caminos son siempre lineales. Las únicas bifurcaciones permitidas son las realizadas mediante símbolos de decisión. Los flujos de dos flechas pueden unirse y proseguir por un único camino (Fig. 5.a). Por el contrario, una flecha no puede bifurcarse como se aprecia en la figura 5.b. Esto es lógico, puesto que entonces nos encontraríamos con situaciones ambiguas. En efecto, en un caso como el de la figura 5b, ¿En qué dirección proseguirá el flujo? En un proceso, las tareas se realizan una detrás de otra (secuencialmente) y no es posible realizar varias tareas simultáneamente (en paralelo). Por consiguiente, la situación anterior da lugar a una incoherencia, ya que no tenemos criterio alguno para decidir la dirección que hay que tomar. Por esta razón decimos que las bifurcaciones sólo pueden llevar a término mediante los símbolos de decisión, pues estos sí que poseen un criterio para decidir qué camino tomar.

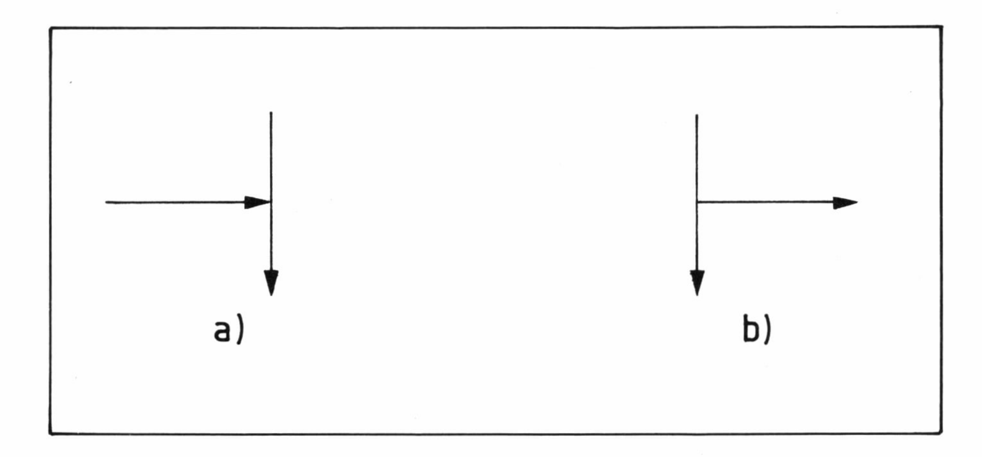


Figura 5: Conexiones de flechas: a) Forma correcta de enlazar el flujo de dos flechas. b) Forma incorrecta de bifurcar el flujo.

7.2.5 Representación gráfica de las instrucciones

Emplearemos ahora el repertorio de símbolos anteriormente visto para el diseño de algunos programas. Pero antes de empezar con el diseño, conviene adquirir una cierta práctica en el manejo de estos símbolos. Comenzaremos realizando el ordinograma para calcular el área de un triángulo. En capítulos anteriores ya construimos un programa para efectuar este cálculo, lo cual nos facilitará ahora la tarea de elaborar el ordinograma (Fig. 6).

En primer lugar, todo ordinograma debe empezar con el bloque de inicio.

A continuación, el proceso deberá leer los valores de base y altura. Por tanto, dispondremos un símbolo de entrada-salida en el cual se leerán los valores de las variables a y b, como se muestra en la figura 6.

El nivel de detalle en un bloque de proceso

A continuación viene la fase de cálculo y emplearemos un bloque de proceso. En este punto conviene hacer algunas precisiones. En un símbolo

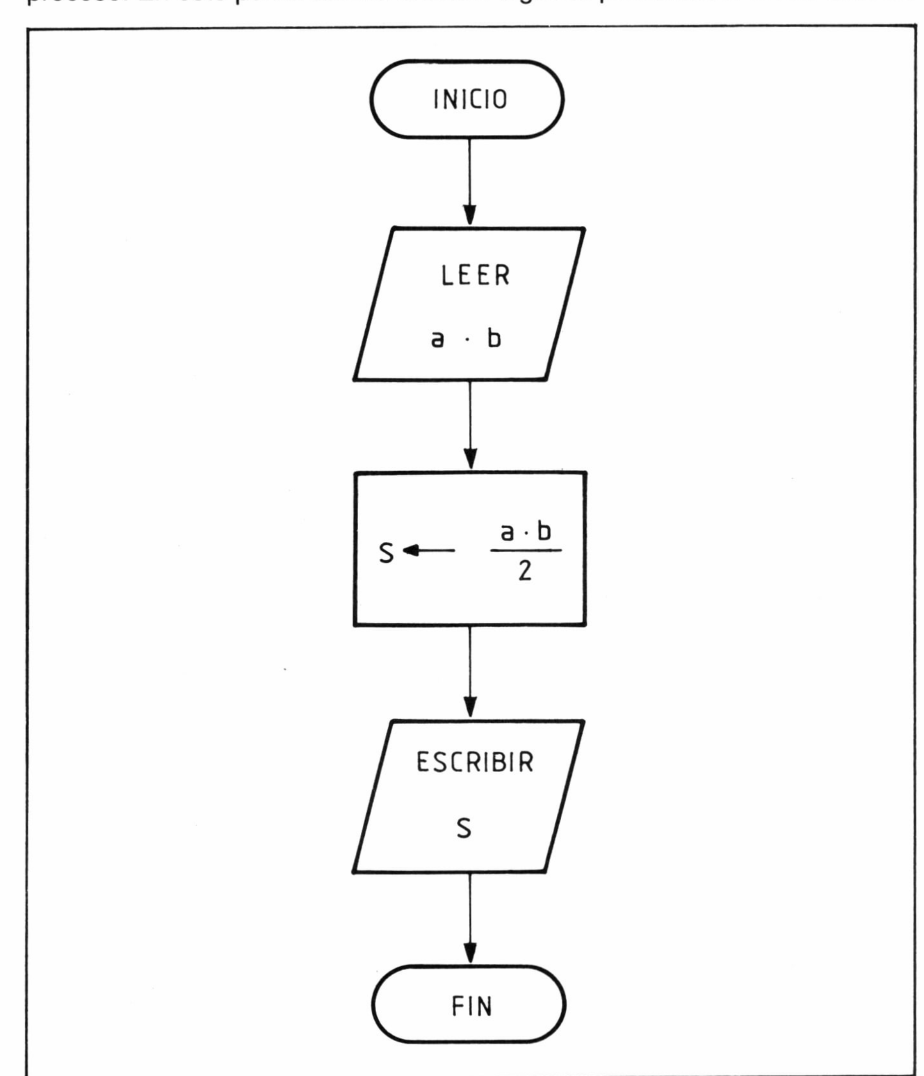


Figura 6: Ordinograma para el cálculo de la superficie de un triángulo.

de proceso se puede emplear un nivel de detalle de acuerdo con las necesidades. Si no nos hubiera interesado el detalle de los cálculos a realizar, hubiéramos colocado un bloque con el texto «Cálculo de la superficie» y esto hubiera bastado. Por el contrario, si queremos especificar claramente estos cálculos, entonces se emplea la fórmula que vemos en la figura 6. En ella se escriben las operaciones de la misma manera que se emplean en Matemáticas. La única diferencia es que se utiliza la flecha para indicar dónde se almacena el resultado; en nuestro caso en S. Seguramente habremos notado ya la gran similitud de este bloque con la instrucción LET del BASIC. Esta similitud es ciertamente intencionada, pues en la mayor parte de lenguajes de programación, y no sólo en BASIC, la forma de escribir las expresiones numéricas es idéntica.

Una vez obtenido el resultado hay que hacerlo visible, para lo cual emplearemos el bloque de salida en donde escribiremos el valor de S.

Finalmente, en los ordinogramas se especifica siempre dónde finalizan mediante el símbolo terminal. Este símbolo podría parecernos innecesario, pero hay que tener en cuenta que muchas veces los ordinogramas poseen más de un punto final y entonces los símbolos terminales son imprescindibles para señalarlos con claridad.

Ahora que hemos terminado el ordinograma tal vez tengamos una cierta sensación de inutilidad, pues podríamos haber escrito el programa directamente. Sin embargo, no debe olvidarse que en este caso hemos partido de una situación inversa a la normal. El programa en BASIC ya estaba hecho (y además era muy sencillo). Por tanto, la labor de diseño, que en realidad hay que hacerla previamente al programa, carecía ya de sentido. El objetivo de este ejemplo era familiarizarnos con el manejo de los distintos símbolos. A partir de aquí, utilizaremos los ordinogramas como lo que realmente son: una herramienta de diseño.

Hemos comentado en un apartado anterior, que un ordinograma podía traducirse fácilmente a cualquier tipo de lenguaje de ordenador. Esta traducción a un lenguaje se suele denominar también *Codificación.* Como ilustración de este punto daremos los listados de este programa en los lenguajes BASIC, FORTRAN, PASCAL y C.

La versión en BASIC ya la conocemos y es:

```
10 REM Programa area
20 INPUT A
30 INPUT B
40 LET S=A*B/2
50 PRINT "AREA=";S
```

En FORTRAN, el listado es:

El mismo ordinograma en distintos lenguajes de programación

```
      PROGRAM AREA
      REAL A,B,S
      READ(*,100) A,B
      S=A*B/2.0
      PRINT *,'AREA=',S
100   FORMAT(2F10.3)
      END
```

El listado en PASCAL es el siguiente:

```
var a,b,s: real;
begin
  read(a,b);
  s:=a*b/2;
  write('area=',s); writeln;
end;
```

Finalmente, en C el programa sería:

```
/* Area triangulo */
main()
{
  float a,b,s;
  getfmt("%f %f",&a,&b);
  s=a*b/2;
  putfmt("area=%f \n",s);
}
```

Aunque la forma de escribirlo o sintaxis es distinta en cada lenguaje, la estructura básica es idéntica en todos ellos, pues esta estructura es la que se ha obtenido haciendo el diseño, que es independiente del lenguaje. Podemos apreciar también la simplicidad sintáctica del BASIC frente a los demás lenguajes.

RESUMEN

La instrucción INPUT sirve para entrar datos, pero en ciertas ocasiones tienen dos inconvenientes: El primero es que hay que pulsar la tecla de fin de línea y el segundo que detiene el programa hasta que contestamos.

La función INKEY$ es una función sin argumentos y cuyo resultado es un texto de longitud cero o uno.

Esta función permite soslayar los inconvenientes del INPUT, ya que consulta en el teclado qué tecla hemos pulsado, retornando el símbolo pulsado si efectivamente se ha pulsado o el texto vacío si no se ha pulsado ninguna tecla.

Para conseguir que la función INKEY$ se espere, es preciso hacer la pregunta de si el texto es vacío y reciclar a la función si efectivamente lo es. Cuando no lo sea, es que se ha pulsado una tecla.

La función INKEY$ devuelve cualquier tecla que se pulse aun cuando no corresponde a caracteres imprimibles; la instrucción INPUT ignora estos caracteres.

La función INKEY$ se utiliza en programas de funcionamiento interactivo; es imprescindible en programas de juegos. Se utiliza con frecuencia para la selección de «Menús». El Menú significa, en el entorno informático, la elección de una de las opciones válidas del programa. Se elimina con la función INKEY$ la necesidad de tocar dos teclas.

Diseñar un programa significa planificar las órdenes y su flujo que hay que dar al ordenador.

El diseño no debe hacerse en un lenguaje de programación; es necesario atender los aspectos básicos de la cuestión, olvidando los detalles.

En el diseño, la semántica prevalece sobre la sintaxis. La semántica se refiere al significado de las cosas; en cambio, la sintaxis se refiere a cómo están escritas.

El diseño se realiza en forma jerárquica, es decir, en una primera fase el problema se divide en subproblemas con un nivel de explicación de las operaciones grosero, en el sentido de poco detalle. Después, cada subproblema se divide, a su vez, en subsubproblemas, explicados cada vez con mayor detalle.

La técnica de diseño con el ordinograma tiene dos funciones importantes: La primera, de ayuda al desarrollo y la segunda, como documentación de un programa.

Los símbolos de los ordinogramas, que a veces se denominan bloques son:

a) La flecha que indica el flujo de proceso.

b) El rectángulo que representa un proceso en general. Permite la descripción a diversos niveles de detalle.

c) El rectángulo inclinado que representa los procesos de entrada y salida.

d) Rectángulo con bordes redondeados, indica el inicio o fin del flujo.

e) El rombo que representa a la toma de decisiones. Es el único símbolo que permite la bifurcación del flujo de instrucciones.

f) El círculo que representa el conector de página y permite establecer conexiones entre partes de ordinogramas que no están en la misma página de papel.

La traducción de un ordinograma a un lenguaje de programación se denomina codificación. La sintaxis es diferente en cada lenguaje, pero la organización de fondo es la misma en todos ellos.

EJERCICIOS DE AUTOCOMPROBACION

Complete las frases siguientes

1. Cuando se utiliza la instrucción INPUT el programa se .... hasta que ha realizado la instrucción.

2. La función .... permite saber cuál es la tecla que se ha pulsado sin detener el programa.

3. Cuando no se ha pulsado ninguna tecla la función INKEY$ devuelve la cadena ....

4. Cuando se pulsa una tecla de carácter no imprimible, la función INKEY$ devuelve una cadena que contiene el .... de la tecla pulsada.

5. Un .... es la manera como se denomina, en un entorno informático, la selección de las opciones válidas para el programa.

6. La planificación de un problema en el ordenador se denomina .... del programa.

7. El aspecto más importante de un diseño es atender a los aspectos .... y olvidar los detalles.

8. El diseño se realiza en forma ....

9. El ordinograma de un programa tiene dos funciones: La de ayudar al desarrollo y como .... del programa.

10. La .... es la traducción del ordinograma a un lenguaje de programación.

Encierre en un círculo la letra que correspondan a la alternativa correcta.

11. El símbolo que se utiliza para iniciar un programa dentro de un ordinograma es:

    a) El rombo.
    b) El rectángulo con bordes redondeados.
    c) El rectángulo inclinado.
    d) El rectángulo.

12. El símbolo que se utiliza para un proceso en general en un ordinograma es:

    a) El rombo.
    b) El rectángulo con bordes redondeados.
    c) El rectángulo inclinado.
    d) El rectángulo.

13. El símbolo que se utiliza para la toma de decisiones en un diseño con ordinograma es:

    a) El rombo.
    b) El rectángulo con bordes redondeados.
    c) El rectángulo inclinado.
    d) El rectángulo.

14. El orden con que se realizan las operaciones, es decir, el flujo del programa se simboliza en los ordinogramas mediante el símbolo:

    a) El rombo.
    b) El rectángulo con bordes redondeados.
    c) La flecha.
    d) El rectángulo.

15. El círculo es un símbolo que se utiliza en los ordinogramas para significar:

    a) Una toma de decisiones.
    b) Una conexión a otra página de papel.
    c) Un proceso de entrada y salida.
    d) El final de un programa.

16. El rectángulo inclina se utiliza en los ordinogramas como símbolo para representar.

    a) El flujo del programa.
    b) La toma de decisiones.
    c) El inicio del programa.
    d) Un proceso de entrada o salida de información.

17. ¿Qué símbolo de los siguientes no permite ninguna entrada de flujo?

    a) El fin del programa.
    b) El proceso en general.
    c) La toma de decisiones.
    d) El inicio del programa.

18. ¿Qué símbolo de los siguientes permite más de una salida?

    a) El fin del programa.
    b) El proceso en general.
    c) La toma de decisiones.
    d) El inicio del programa.

19. Las flechas que indican el flujo no pueden

    a) Unirse para formar una.
    b) Desviarse a otra página del ordinograma.
    c) Desdoblarse en dos ramas de flujo.
    d) Reciclarse al inicio del programa.

20. La tecla INKEY$ debe utilizarse en programas de

    a) Cálculo matemático.
    b) Tipo interactivo.
    c) Gestión comercial.
    d) Enseñanza por ordenador.

7.3 CASOS PRACTICOS

Mediante algunos casos prácticos estudiaremos la manera cómo se utiliza un ordinograma para elaborar el diseño de un programa. En la lección de prácticas realizaremos la traducción a lenguaje BASIC de cada uno de los ordinogramas.

7.3.1 Ecuación de segundo grado

Recordemos brevemente los conceptos fundamentales referentes a una ecuación de segundo grado.

Si no recuerda lo que es la ecuación segundo grado, no se preocupe demasiado, habrá ciertos aspectos de este caso práctico que no acabará de comprender. Sin embargo, no debe fijarse demasiado en los aspectos matemáticos del problema sino más bien en los aspectos de programación que sí puede entender. Cuántas veces los programadores profesionales deben realizar programas sobre temas de los que no entienden. Ciertamente la comprensión a medias de este caso no le impedirá continuar con el curso.

La forma general de esta ecuación es:

$$a\,x^2 + b\,x + c = 0$$

en donde, x^2 significa x*x.

Las ecuaciones de segundo grado tienen dos soluciones que se denominan raíces.

Las fórmulas para calcular las soluciones son:

$$r_1 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \qquad r_2 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

Identificación de a, b, c y hallar las soluciones

Por ejemplo, si tenemos la ecuación $2x^2 - 3x + 1 = 0$, debemos saber identificar la *a* como 2, la b como −3 y la c como 1. Las soluciones serán el resultado de sustituir los valores a, b y c identificados en las fórmulas de más arriba; así sustituyendo los valores en las fórmulas obtenemos:

$$r_1 = \frac{-(-3) - \sqrt{9\text{-}4x2x1}}{2x2} = 0{,}5 \qquad r_2 = \frac{-(-3) + \sqrt{9-4x2x1}}{2x2} = 1$$

Se puede comprobar fácilmente que si sustituimos los valores hallados, 0,5 y 1 en la ecuación original, se verifica la igualdad

$$2x0{,}5^2 - 3x0{,}5 + 1 = 0{,}5 - 1{,}5 + 1 = 0$$

$$2x1^2 - 3x1 + 1 = 2 - 3 + 1 = 0$$

Supongamos ahora que nos encargan la tarea de programar el ordenador para que resuelva ecuaciones de segundo grado.

Empezaremos utilizando lo que hemos aprendido sobre diseño jerárquico y, en primer lugar, realizaremos un ordinograma que tenga en cuenta

únicamente los aspectos básicos del problema, dejando los detalles para más adelante. La primera ocasión que deberá hacer el programa será la lectura de los datos referentes a la ecuación en concreto que queremos resolver. En una segunda fase se realizarán los cálculos para obtener las soluciones que finalmente se escribirán. El ordinograma que construimos para representar este diseño previo se muestra en la figura 7.

Como vemos, en cada bloque se han colocado textos explicativos en lugar de especificar las operaciones. La utilidad de este diseño previo radica en que nos ayuda a tener una visión global del mismo y, lo que es más importante, a separar el problema general (resolver una ecuación de segundo grado) en problemas más pequeños y por consiguiente, más sencilla.

Antes de seguir adelante con el diseño hay que analizar un punto importante sobre las ecuaciones de segundo grado. Ocurre que no siempre existe una solución real de las mismas. Por ejemplo, si intentamos resolver la ecuación $2x^2 + 4x + 5 = 0$, veremos que al aplicar la fórmula, nos queda la raíz cuadrada de un número negativo que, como sabemos, no se puede calcular. Esto no quiere decir que la ecuación no tenga solución, sino que ésta no es de tipo real. Para resolver las raíces cuadradas de números negativos hay que acudir a los números imaginarios y entonces la solución de la ecuación de segundo grado es de tipo complejo (con parte real y parte imaginaria). Sin embargo, estos aspectos matemáticos están alejados de nuestros objetivos, pero hay un punto que no podemos olvidar. Se debe controlar que en el cálculo de la raíz cuadrada, el número de su interior no sea negativo puesto que entonces el ordenador nos daría un error.

El control de si se puede evaluar la solución lo efectuaremos mediante un bloque de decisión. El ordinograma que representa este diseño intermedio se muestra en la figura 8. De las tres partes del ordinograma inicial, la primera es la entrada de datos. Está claro que los datos que necesitamos para resolver una ecuación, son los valores a, b y c. Por tanto, el bloque de lectura de datos se ha cambiado por uno que especifica que se leen a, b y c.

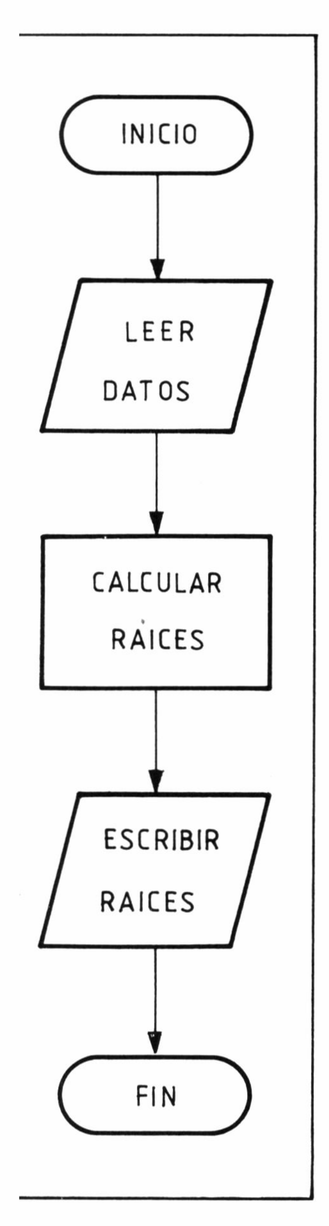
Desglose del bloque de cálculo

El desglose del bloque de cálculo es más complicado. Antes de evaluar las raíces hay que averiguar si este cálculo es posible.

Para ello disponemos un bloque de decisión que determina si la ecuación tiene una solución real. Si no es así, se escribe un mensaje explicativo y se termina el proceso. Si la solución es real, se pasa a la fase de cálculo de las raíces y posteriormente se escriben, finalizando el proceso.

La parte de lectura y de escritura ya están suficientemente detalladas. Queda, por fin, la cuestión de cómo se realizará la pregunta sobre si la solución es real o compleja. La única manera de saberlo es haciendo las operaciones que están incluidas dentro de la raíz cuadrada. Entonces el bloque de decisión consistirá en determinar si este número es negativo o positivo. Dicho de otra manera, la pregunta será si es menor que cero o no. Si lo es, quiere decir que la solución es compleja y entonces no se efectuarán los cálculos y en cambio se escribirá en mensaje informativo. Por el contrario, se calculará cada una de las soluciones con las fórmulas citadas al principio.

El ordinograma que corresponde a esta última fase del diseño se muestra en la figura 9. Este ordinograma ya es el definitivo y se puede traducir directamente a un lenguaje de programación.



INICIO

LEER

a · b · c

¿SOLUCION REAL?

No

Si

CALCULAR

r_1 y r_2

ESCRIBIR

r_1 · r_2

FIN

ESCRIBIR "SOLUCION COMPLEJA"

FIN

igura 7: Ordinograma con el iseño previo para el cálculo de cuaciones de segundo grado.

igura 8: Ordinograma con el iseño intermedio para el cálculo e ecuaciones de segundo rado.



Este ordinograma es un ejemplo de proceso con dos puntos terminales. Insistimos en el hecho de que el flujo es siempre único y la bifurcación que se realiza en el bloque de decisión es exclusiva, es decir, que el flujo sigue por un camino o por el otro, pero nunca por ambos a la vez.

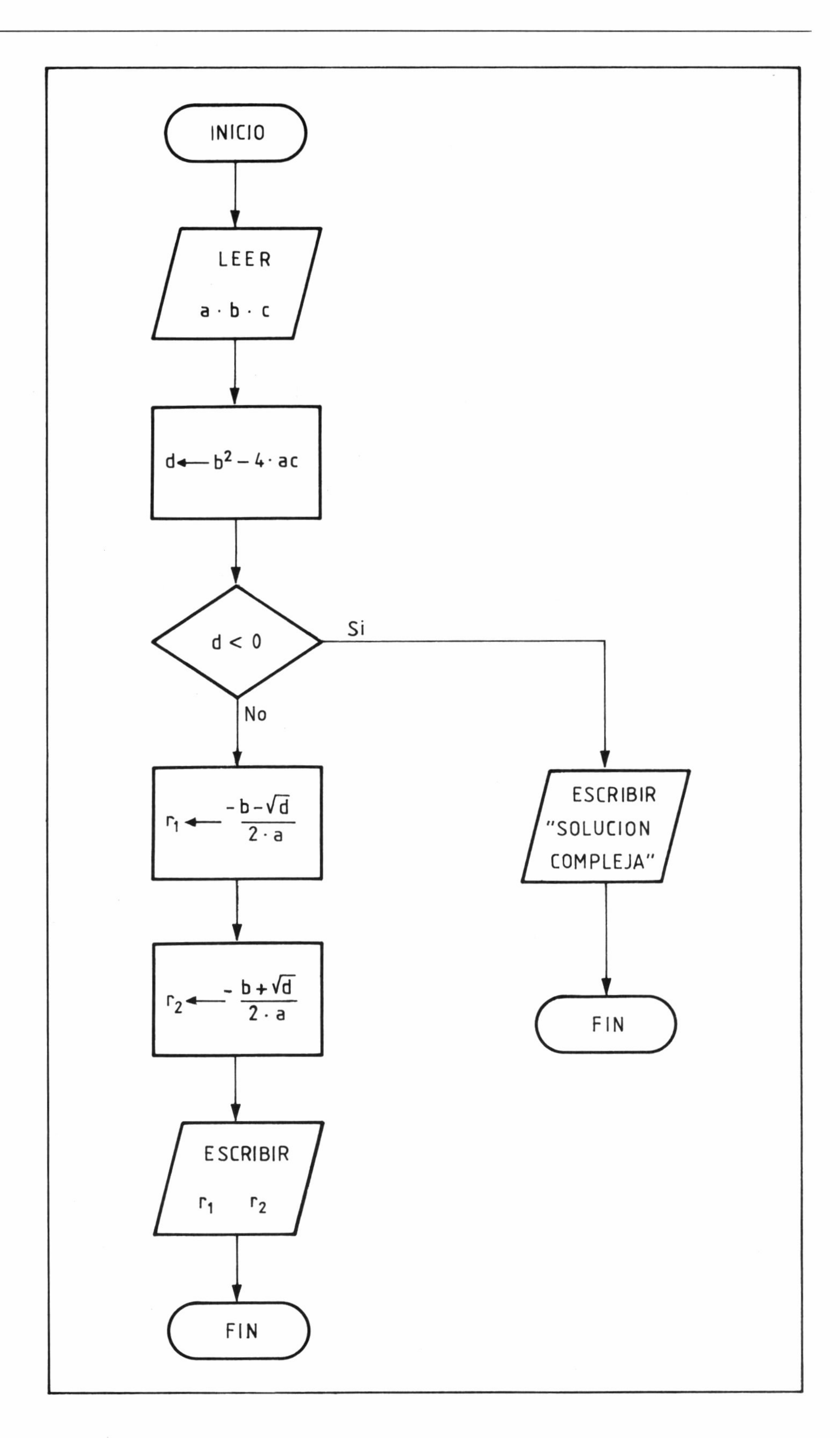


Figura 9: Diseño final para el cálculo de ecuaciones de segundo grado.

7.3.2 Actualización de precios

Se trata de construir un ordinograma para resolver la siguiente cuestión. Se dispone de una lista de precios de productos y se les quiere aplicar un incremento (igual para todos) a fin de obtener una nueva lista con los precios actualizados.

En este caso hay dos tipos de entrada de datos bien diferenciados: En el primero de ellos, se lee el tanto por ciento de incremento. En el segundo, se lee el precio de un producto.

Esta separación es necesaria, puesto que el incremento sólo hay que leerlo una única vez ya que es el mismo para todos los productos. Por el contrario, la lectura del precio se repetirá para cada uno de los productos.

Cuando se ha leído el precio, se pasa a la fase de cálculo del precio actualizado, que finalmente se escribe. En la figura 10 se representa el ordinograma correspondiente a este diseño. El tanto por ciento de incremento se representa por la letra T, el precio por la letra P y la R representa el precio actualizado. En él, se ve que una vez escrito el nuevo precio, el control se dirige de nuevo a la lectura de otro precio antiguo para el cual se repetirán los cálculos.

Es necesario buscar la condición de final

Observando con detenimiento este ordinograma, vemos que le falta un punto importante. No tiene el bloque terminal que indique dónde acaba el proceso. Por tanto, tal como está ahora, el proceso no tiene fin y la máquina insistiría siempre en pedirnos de nuevo otro precio para realizar los cálculos. En principio esto no es problema, puesto que se trata de un diseño previo, en el que solamente nos interesaba el planteo general de la solución.

Se ha diseñado un programa que no tiene fin

Para la siguiente fase del diseño, hay que buscar una solución a este punto. Es necesario encontrar un sistema para indicar a la máquina que debe acabar el proceso. La forma más normal de hacerlo es establecer un dato particular, que servirá para que la máquina lo tome como condición de final. En nuestro ejemplo, esta condición podría ser cuando el precio final fuera igual a cero. Esta elección es muy acertada puesto que no tiene sentido aplicar un incremento a un precio que vale cero. Entonces haremos que la máquina considere que este precio igual a cero sea la indicación de que debe finalizar el proceso. En el diseño final colocaremos un bloque de decisión justo después de la lectura del precio. En él haremos precisamente la pregunta de si P vale cero. Si la respuesta es negativa, el flujo seguirá hacia la fase de cálculo tal como lo hacía antes. Si la respuesta es afirmativa, es decir, P vale cero, el flujo se dirige a un símbolo terminal. De este modo queda establecido el punto por donde finaliza el ordinograma.

Queda, por último, detallar las operaciones que deben efectuarse para calcular el nuevo precio. En realidad son muy sencillas. La cantidad que hay que sumar al precio antiguo es P*T/100. Por tanto, el nuevo precio R será:

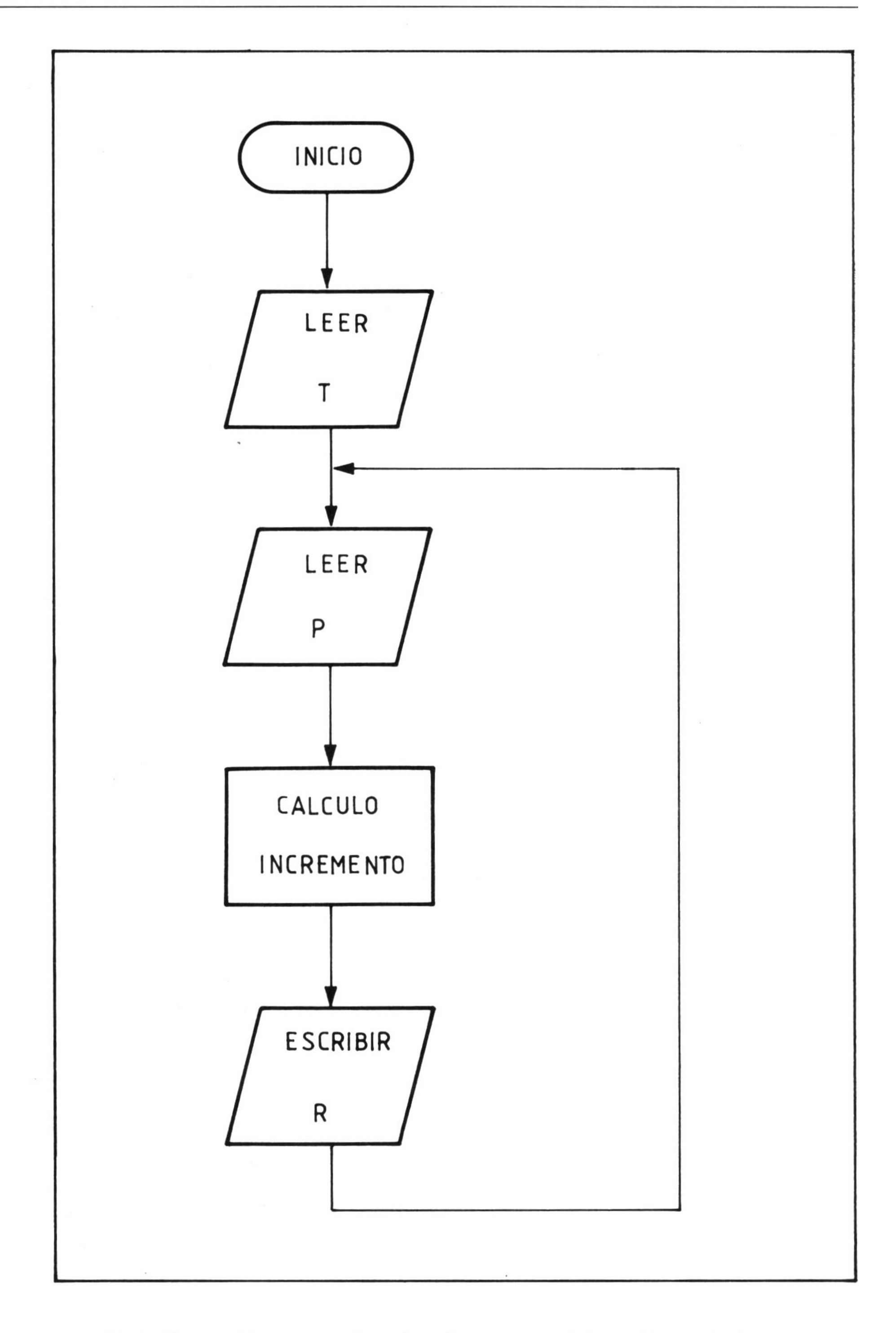
R=P+P*T/100

o bien, agrupando términos

R=P*(1+T/100)

Esta operación es la que colocaremos en el bloque de cálculo.

Figura 10: Diseño inicial para el programa de actualización de precios.



En la figura 11 se muestra el ordinograma elaborado según lo que acabamos de explicar. Al tratarse de un diseño final, puede ya traducirse directamente a BASIC. Es interesante que intentemos realizar esta codificación por nosotros mismos antes de consultar el capítulo de prácticas. Un punto a tener en cuentra es que la flecha que, después de escribir R, retro-

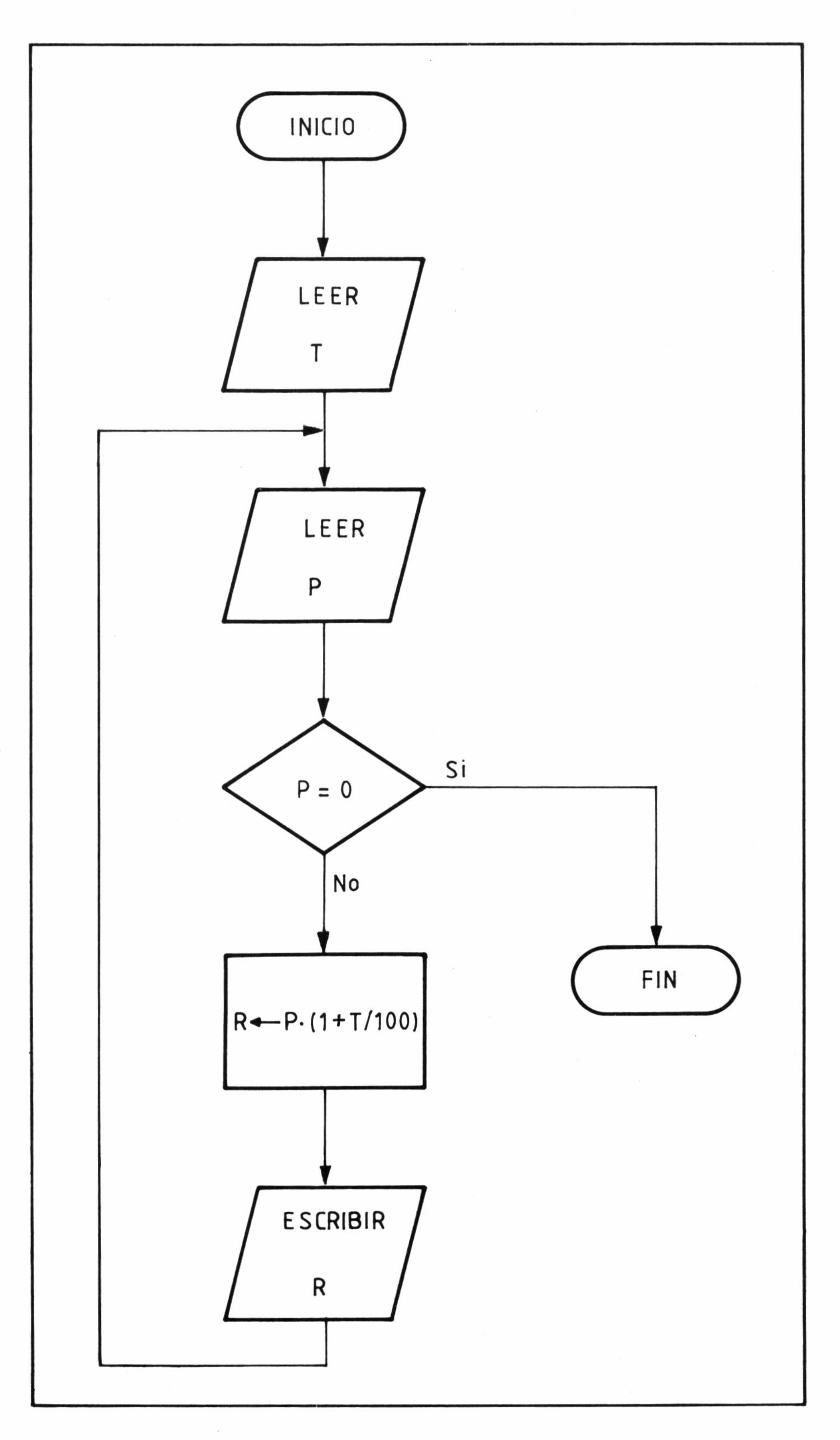


Figura 11: Ordinograma correspondiente al diseño final para el programa de actualización de precios.

cede a la fase de lectura hay que traducirla en BASIC por la instrucción GOTO.



7.3.3 Números pares

En este ejemplo, construiremos un ordinograma para determinar si un número es par o impar. El funcionamiento será el siguiente: Le entraremos un número al ordenador y éste nos responderá si dicho número es par o impar.

Según este planteo, el diseño previo consistirá en un bloque de entrada donde se leerá el número. A continuación vendrá la determinación de si el número es par o no. Este bloque tendrá dos salidas y ambas se dirigen a bloques de escritura. En uno de ellos se escribirá un mensaje indicando que el número es par y en el otro mensaje indicando que es impar. Después de los dos símbolos de salida, el proceso finaliza. El ordinograma que refleja este planteo se muestra en la figura 12.

Diseño de la decisión en forma jerárquica

Nos queda un punto crucial. ¿Cómo determinar si un número es par? En el capítulo dedicado a estudiar las funciones aritméticas, vimos la función INT que suprimía los decimales. Esto nos permitiría saber si un número era múltiplo de otro. En este caso podríamos aplicarla para ver si es múltiplo de dos. No obstante, vamos a suponer que, por las causas que sean, no disponemos de la función INT. Una forma de resolver el problema sería restar 2 al número en cuestión, sucesivas veces hasta llegar a cero o a uno. Notemos que si vamos restando repetidamente 2, al final llegaremos a 1 si el número era impar y a cero si el número era par. Si hacemos la prueba manualmente para algunos números pequeños veremos fácilmente como se cumple la afirmación anterior. Pues bien, este sistema lo emplearemos en nuestro programa. Hay que advertir que este procedimiento de resta, lo efectuamos puramente como ejercicio académico y en la práctica se utilizaría la función INT.

El bloque de decisión anterior lo subdividiremos en varias partes. Para empezar, construiremos el proceso de restas sucesivas. Está claro que se necesitará colocar un bloque de cálculo, en el cual se decrementará en dos el número inicial N. Este paso se repetirá mientras N sea mayor que 1. Para controlar este punto situaremos un símbolo de decisión donde determinaremos esta condición. Si N es mayor que 1, lo enviaremos a efectuar otra vez la resta. Finalmente, después de varias restas, N ya no será mayor que 1 y entonces será el momento de determinar si es par o no.

Cuando decimos que N no es mayor que 1, significa que vale o bien 1 ó bien 0. Tal como realizamos las restas, no es posible ningún otro resultado.

A continuación colocaremos un bloque de decisión. Dentro, preguntaremos si N es cero. En caso afirmativo, dirigiremos el flujo hacia un símbolo de salida que escribirá el mensaje «ES PAR». Si por el contrario, N no es cero, dirigiremos el flujo hacia otro símbolo de salida que escribirá el mensaje «ES IMPAR».

En la figura 13 se representa el ordinograma construido según este diseño. Un aspecto importante a resaltar es que el bloque de decisión general del diseño previo (Fig. 12) se ha subdividido en el diseño final, en dos

INICIO

LEER N

¿ES PAR?

Si

No

ESCRIBIR "PAR"

ESCRIBIR "IMPAR"

FIN

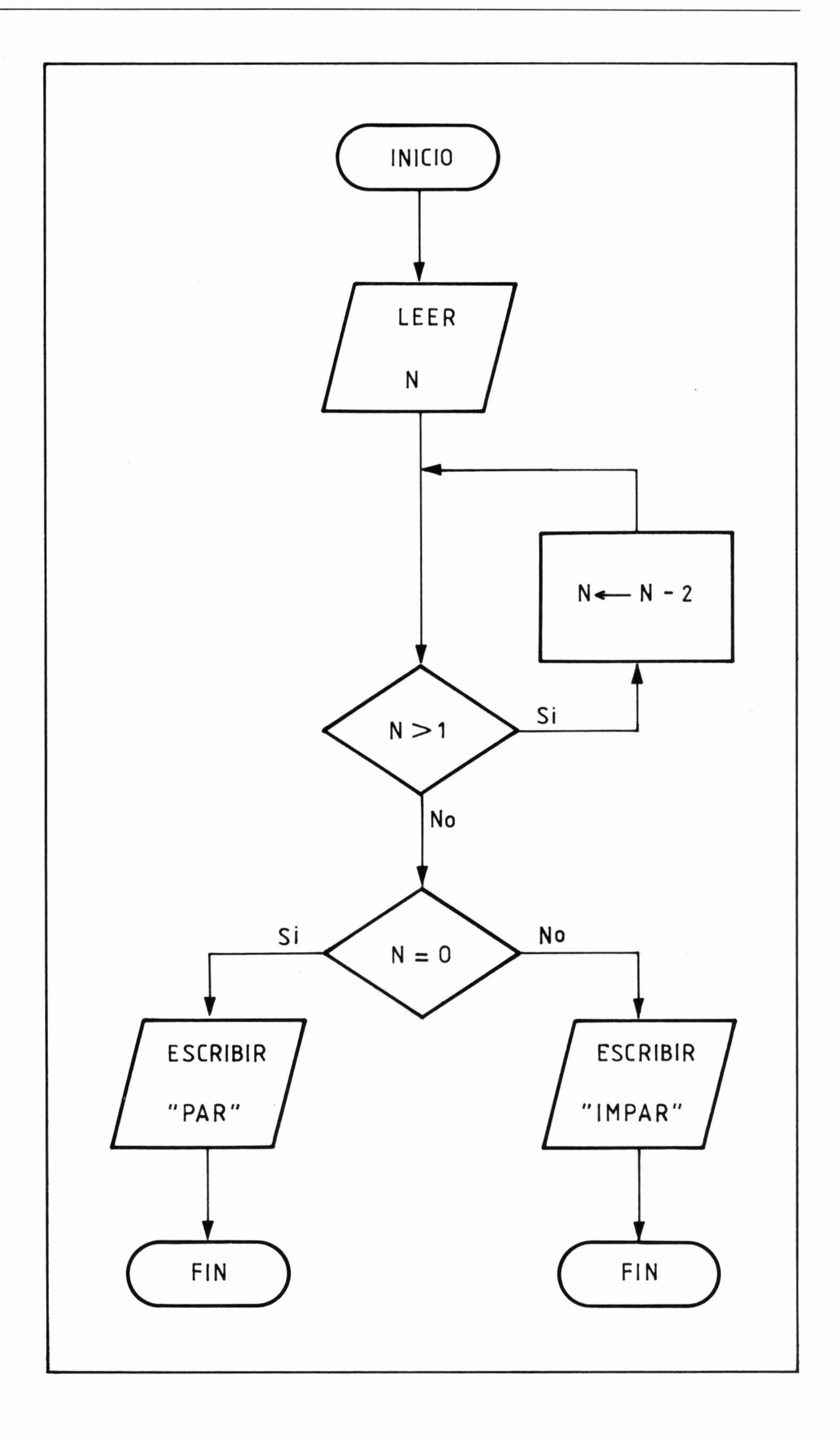
FIN

Figura 12: Planteo inicial para determinar si un número es par.

bloques de decisión más un bloque de proceso. Este punto muestra con claridad que la técnica de diseño jerárquico se aplica no sólo a los símbolos de proceso, sino también a los demás. En un primer momento ha sido suficiente colocar un bloque de decisión con una pregunta de tipo general. En cambio, en el diseño final se han detallado todos los pasos que indica esta pregunta. Es por esta razón que no hemos utilizado la función INT, pues con ella no se.podía subdividir tanto el bloque de decisión. El sistema de restas nos ha sido más útil como ejercicio de diseño, aunque desde el punto de vista del funcionamiento sea mucho menos eficiente que la función INT.



Figura 13: Ordinograma completo para determinar si un número es par.



7.3.4 Interés compuesto

Como último ejemplo, construiremos un ordinograma para calcular el valor del capital al cabo de un cierto tiempo, aplicando el interés compuesto. La fórmula del interés compuesto es:

$$F=C\ (1+R/100)^T$$

donde F es el capital final, C es el capital inicial, R es el rédito expresado en tanto por ciento y T es el período de tiempo. Esta fórmula nos da el campital final directamente después de T años. Por ejemplo, si el capital inicial son 2.000.000, el rédito es de un 10 % anual y T son 2 años, el capital final será

$$F=2000000(1+10/100)^2 = 2420000$$

Si lo hubiésemos evaluado paso a paso, los cálculos serían los siguientes. Para el primer año, los intereses serían un diez por ciento de dos millones, es decir, 200.000. El capital final es pues de 2.200.000. Para el segundo año, los intereses serán otro diez por ciento del último capital, o sea 220.000 que sumados al capital resulta un valor de 2.420.000 que coincide con el calculado por la fórmula.

El ordinograma que elaboramos nos escribirá el valor del capital para los cinco primeros años. Puesto que el problema no es demasiado difícil y ya conocemos bastante bien cómo construir ordinogramas, empezaremos directamente por el diseño final.

Variables intermedios para el cálculo. No se leen como dato

En primer lugar, los datos que necesita la máquina son el valor del capital inicial y el rédito. Por tanto, empezaremos colocando el bloque de lectura de las variables C y R. El valor de T se calculará internamente, e irá variando de 1 a 5. Para cada valor de T se evalúa la fórmula. Para representar este cálculo dispondremos un bloque de proceso en el ordinograma. Seguidamente situaremos un símbolo de salida para indicar que se escribe el valor de F. Al final se coloca un bloque de decisión para determinar si se ha completado el proceso. Si no es así, se incrementa el valor de T y se repite el cálculo. En la figura 14 se muestra el ordinograma que refleja este proceso. Fijémenos que debe asignarse un valor a la variable T antes de utilizarla para calcular F. De no hacerlo así, el planteamiento hubiera sido erróneo, pues en el momento del cálculo, el valor de T hubiese sido indefinido. Por otra parte, no hay que leer T en el bloque de entrada, puesto que su valor está definido por el propio proceso y el operador no puede modificarlo.

La flecha que cierra el lazo está situada justo debajo de la asignación inicial de T. Si los dos flujos se unieran por encima de este bloque de proceso, se calcularía siempre el mismo valor del capital (el primer año) y el programa no tendría fin.

Si le entramos un capital inicial de 2.000.000 y un rédito de un 10 % anual, los cinco resultados escritos serán

2200000	2662000	3221020
2420000	2928200	

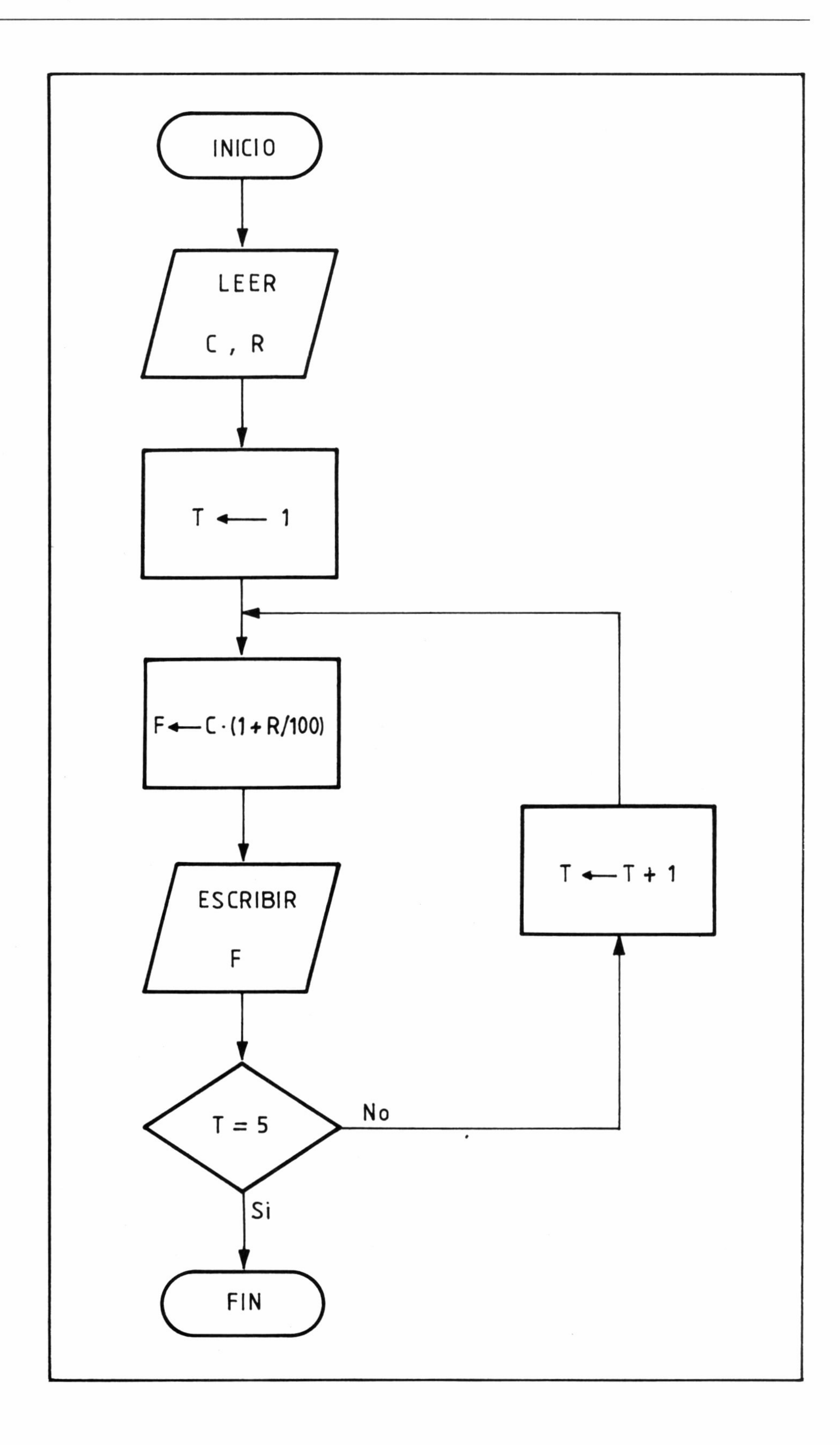
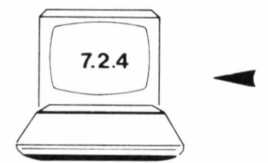


Figura 14: Ordinograma para calcular el capital final, aplicando el interés compuesto.

que corresponden al capital final al cabo de un año, dos años, etc.; hasta cinco años. En la mayoría de estos ordinogramas hemos empleado procesos de repetición. La importancia de este tipo de procesos es enorme en informática y por ello dedicaremos más adelante un capítulo completo a estudiarlos.



7.4 CODIFICACION

Cuando llega el momento de traducir el ordinograma o diagrama de flujo a un lenguaje (fase de codificación) hay que tener presente una serie de normas.

La lista de variables

En primer lugar debemos elaborar una lista de variables que aparecen a lo largo del ordinograma. Esta lista incluirá, además del nombre de la variable, una breve descripción de la misma, así como también el tipo de dato (numérico o textual) que contiene. Si hay restricciones en los valores que puede tomar la variable conviene incluirlos también en la lista. Por ejemplo, que la variable tiene siempre valores positivos o que contendrá un número entre 1 y 10, o cualquier otra restricción o intervalo de validez.

Los ejemplos que hemos visto hasta este momento son muy cortos y parece innecesario construir esta lista. Sin embargo, no olvidemos que en la práctica no son raros los programas con varios cientos de instrucciones. Entonces, la lista nos ayuda a recordar qué dato contiene cada variable y, sobre todo, a no dar nombres repetidos a distintas variables. Además, los intervalos de validez nos ayudarán en la fase de puesta a punto. Si el programa no funciona correctamente, consultaremos los valores de las variables y comprobaremos si están dentro del margen aceptado.

Por otra parte, queda la cuestión de cómo se traducen los símbolos del ordinograma a las correspondientes instrucciones. Le recomendamos que preste mucha atención a lo que viene pues le ahorrará mucho tiempo a la hora de escribir los programas.

En BASIC, los símbolos de inicio no se traducen, puesto que la numeración de las listas ya indica por sí misma donde empieza el programa. El símbolo terminal sólo hace falta traducirlo por un END o un STOP si hay más de uno. Si no, la propia numeración establece automáticamente el final. Los símbolos de proceso se traducen por la instrucción LET y únicamente se debe prestar atención en escribir las expresiones según las normas del lenguaje, es decir, respetando el orden de prioridad de los operadores.

Correspondencia entre símbolos e instrucciones

El bloque de decisión se traducirá por una instrucción IF y un GOTO al número de línea correspondiente. El equivalente de la segunda flecha del bloque de decisión será la instrucción situada inmediatamente debajo de la instrucción IF. Ya sabemos que en un IF, cuando la condición es falsa, el programa sigue por la instrucción situada a continuación. Esta afirmación no se cumple si utilizamos una instrucción IF..THEN..ELSE. En ese caso, la segunda flecha corresponde a la instrucción que sigue a ELSE. A veces se pueden englobar dos bloques de decisión en un solo IF mediante la utilización de operadores Booleanos (AND, OR...).

Cuando la línea de flujo (la flecha) retroceda o avance a saltos sin se-

es la comparación del programa contenido en memoria con el programa contenido en la cinta magnética.

Dado, pues, que se ha de leer lo que hemos grabado, antes de realizar la verificación, deberemos rebobinar la cinta hasta un poco antes del punto en que hemos iniciado la grabación.

Si se detecta que hay diferencias entre el programa grabado y el programa existente en memoria, la máquina dará el correspondiente mensaje, indicando que han habido errores en la grabación. El tipo de mensaje depende una vez más del modelo de ordenador, y lo encontrará explicado en el correspondiente capítulo de prácticas.

El mecanismo es parecido al LOAD pero sin modificar la memoria del ordenador

Si la grabación ha sido correcta —no hay discrepancias— algunas máquinas lo indican dando un mensaje y otras devuelven simplemente el control al usuario sin indicar nada más. Es el momento de detener el movimiento de la cinta. En caso contrario habrá que repetir de nuevo la grabación, asegurándose de que tiene el sistema bien conectado, de que en la cinta no haya nada grabado (al menos en la zona en que se inicia la grabación del programa) y de que el cassette está ajustado correctamete. Una vez grabado deberá verificar de nuevo, repitiendo el proceso hasta que el resultado de la verificación sea el correcto.

Este comando puede utilizarse también para verificar datos grabados, en los modelos de ordenador que lo permiten. Para conocer en este caso la sintaxis de comando, consulte el capítulo de prácticas.



8.5 OTRA ORDEN DE LECTURA: MEZCLAR PROGRAMAS

Hemos visto ya los dos comandos que nos permiten almacenar un programa de forma permanente en una cinta magnética. Vamos a ver ahora los nuevos comandos que nos servirán para leer —cargar en la memoria del ordenador— los programas que se hayan grabado en una cinta de cassette.

El comando LOAD nos permite, tal como se ha visto, cargar en la memoria del ordenador, un programa grabado en una cinta magnética.

Hemos dicho también, que al utilizar este comando para leer un programa, se pierde el contenido de la memoria, siendo sustituido por el nuevo programa que haya sido cargado.

Existe sin embargo otro comando para leer programas, cuyos efectos son los mismos que el comando LOAD en el aspecto de lectura y control del cassette, pero que difiere de aquél en que no se borra la memoria antes de cargar el nuevo programa. De esta manera se consigue un efecto muy interesante, que es la posibilidad de mezclar programas. Para ejecutar este comando, se escribe la palabra

MERGE

Y a continuación el nombre entre comillas del nuevo programa que se desea cargar. La palabra MERGE significa mezclar, en inglés.

Así, supongamos que tenemos en memoria un programa cualquiera, que llamaremos programa A. Si deseamos cargar otro programa, por ejemplo el programa B, deberemos escribir

Hay dispositivos que memorizan los datos en forma no legible

entrada de datos que ya ha caído practicamente en desuso son las tarjetas o fichas perforables que se representan con el símbolo de la figura 15e.

Todos estos símbolos indican procesos de entrada-salida en los cuales los datos exteriores tienen un formato legible por el usuario, es decir, que los números se han convertido de binario a decimal y las letras se han traducido de ASCII a nuestro alfabeto. No obstante, a veces el ordenador se comunica con el exterior de forma que los datos no son legibles por el usuario. Esto ocurre cuando los datos se leen o se graban en dispositivos magnéticos en los cuales los datos se guardan codificados. Los dos símbolos que indican un proceso de este tipo se muestran en las figuras 15c y 15f. En la primera de ellas, el símbolo representa el acceso a una cinta magnética. El símbolo tiene la forma de un carrete de cinta. El segundo, representa el acceso a un disco magnético.

Se utiliza la cinta magnética cuando hay que leer los datos, uno detrás de otro (acceso secuencial) ya que para efectuar la lectura de una cinta es necesario que ésta se desplace en el sentido de avance. El disco magnético se utiliza para almacenar archivos cuyos datos se leerán en cualquier orden (acceso al azar). Esto es posible porque el brazo del disco se puede desplazar sobre su superficie y colocarse directamente encima de la zona donde se encuentra grabado el dato.

7.6 TABLAS DE DECISION

Las tablas de decisión son un método para la exposición gráfica, estandarizada y simple de un proceso lógico, a través de una representación en forma de tabla.

Cuando se trata de construir un programa para resolver un problema práctico, a menudo aparece una gran cantidad de alternativas y situaciones distintas que hay que tener en cuenta en conjunto. Algunas de estas alternativas se excluyen entre sí; otras, en cambio, serán simultáneas y a cada una le corresponderá una acción distinta. Al elaborar el programa tendremos presente todas las distintas posibilidades que se pueden dar, a fin de asegurar que el programa funcione para todos los casos posibles.

Las tablas de decisión nos servirán para agrupar y sistematizar el conjunto de condiciones y acciones que deben emprenderse.

7.6.1 Formato de la tabla de decisión

La estructura de una tabla de decisión consta de cuatro partes. En la figura 16 se muestra el formato general. En la parte superior izquierda se coloca la lista de condiciones que pueden ser ciertas o falsas. A la derecha (ESTADO DE CONDICIONES), se sitúan los estados de todas las condiciones. En la parte inferior izquierda se escribe la lista de acciones o tratamientos posibles que tendrá el programa. Finalmente, en el recuadro inferior derecho, se establece la relación entre conjunto de condiciones y tratamientos a efectuar.

Figura 16: Formato general de una tabla de decisión.

CONDICIONES	ESTADOS DE LAS CONDICIONES
TRATAMIENTOS	

Presenta de forma clara la relación entre estados y acciones

En el apartado siguiente veremos con más detalle las definiciones de todos estos conceptos. Sin embargo, ya sabemos lo suficiente para construir la primera tabla de decisión.

Empezaremos con un ejemplo muy sencillo. Supongamos que hemos de clasificar a un conjunto de alumnos en aprobados o suspendidos. Diremos que están aprobados si tienen una nota igual o superior a 5, y suspendidos si es inferior. En este caso tenemos únicamente dos condiciones. La primera de ellas es «tener una nota igual o superior a 5» y la segunda «tener una nota inferior a 5». Los estados de estas condiciones sólo pueden ser ciertos o falsos. Tenemos también dos tratamientos o acciones. Una es aprobar y la otra es suspender. En la figura 17 se muestra como quedaría esta tabla de decisión. Fijémonos que se señalan con una cruz aquellos tratamientos que deben realizarse cuando se cumplan las condiciones.

7.6.2 Definiciones

Daremos a continuación una serie de definiciones sobre los conceptos fundamentales.

Regla de decisión

Una *regla de decisión* es una declaración que indica qué conjunto de condiciones deben satisfacerse para que se ejecuten una serie de acciones. Por ejemplo, si el empleado tiene la categoría 1 y trabaja en la sección C y es soltero, entonces el valor de la hora extraordinaria será X. En la tabla de la figura 17 sería: Si la nota es inferior a 5, entonces está suspendido.

Valor de la condición

El *valor de una condición* es el resultado de evaluar el cumplimiento o incumplimiento de una condición. Dos condiciones se llaman *independientes* si los valores que toma una de ellas no dependen de los que tenga la otra. En caso contrario son *dependientes.* Por ejemplo, en el caso de que se dé la condición de aprobado, no puede ser suspendido y viceversa, las dos condiciones son dependientes. En cambio, la condición de que el trabajador es soltero, es independiente de la condición de ser varón. Se dan las cuatro posibilidades, soltero y varón, soltero y hembra, no-soltero y varón y no-soltero y hembra.

Calificación igual o superior a 5	SI	NO
Calificación inferior a 5	NO	SI
Aprobar	X	
Suspender		X

Figura 17: Tabla de decisión para clasificar las notas en aprobado o suspendido.

Condiciones independientes, dependientes exclusivas y dependientes solapadas

A su vez, las condiciones dependientes se subdividen en *exclusivas* y *solapadas*. Las exclusivas son las que no pueden ser ciertas simultáneamente. Por ejemplo, si para preguntar el color de un objeto utilizamos las dos condiciones «Color igual a negro» y «Color igua' a blanco», está claro que no pueden ser ciertas simultáneamente. Las condiciones solapadas son las que sí pueden ser ciertas simultáneamente. Si en el ejemplo precedente, las condiciones fuesen «Color distinto de negro» y «Color igual a blanco», entonces ambas condiciones podrían ser ciertas a la vez.

Así como los ordinogramas nos permiten representar el funcionamiento completo de un programa, las tablas de decisión solamente sirven para clasificar las distintas posibilidades que se pueden presentar. Con las tablas de decisión, establecemos claramente las acciones que deben llevarse a cabo según sea el estado de la lista de condiciones. Por tanto, a partir de una tabla de decisión no podremos construir un programa completo, sino solamente el núcleo fundamental del proceso. Las operaciones de entrada y salida de datos, por ejemplo, no se tienen en cuenta. Podemos decir pues, que las tablas de decisión son una herramienta auxiliar de diseño. Esto no debe llevarnos a minusvalorar su importancia. En aquellos procesos en los que hay varias condiciones simultáneas, su uso es muy recomendable, a fin de asegurar que el programa no deja ninguna posibilidad sin tratar y que, por tanto, funcionará correctamente para todos los casos posibles.

7.6.3 Casos prácticos

La tabla de decisión de la figura 17 se puede ampliar de modo que abarque una clasificación más detallada. Por ejemplo, clasificaremos las notas en muy deficiente, deficiente, suficiente, bien y excelente. En la figura 18 se muestra la tabla de decisión correspondiente a esta clasificación. Fi-

Calificación entre cero y tres	S	N	N	N	N
Calificación entre tres y cinco	N	S	N	N	N
Calificación entre 5 y 6	N	N	S	N	N
Calificación entre 6 y 9	N	N	N	S	N
Calificación 9 o superior a 9	N	N	N	N	S
Muy deficiente	X				
Deficiente		X			
Suficiente			X		
Bien				X	
Excelente					X

Figura 18: Tabla de decisión para clasificar las notas.

jémonos que todas las condiciones son dependientes porque únicamente se tiene en cuenta un valor (la nota). Además, la dependencia es exclusiva por que no pueden ser ciertas simultáneamente.

Las reglas de decisión se clasifican según el esquema de alternativas múltiples

Cuando llega el momento de traducir una tabla de decisión a BASIC (fase de codificación), las instrucciones más importantes serán IF y GOTO. Por ejemplo, para la tabla 17, el programa sería

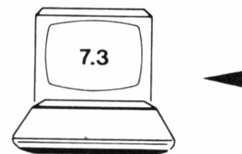
```
10 INPUT N
20 IF N<5 THEN GOTO 50
30 PRINT "APROBADO"
40 GOTO 60
50 PRINT "SUSPENDIDO"
60 STOP
```

En este caso, la traducción es casi inmediata, pero si el número de condiciones aumenta, el programa se complica. Aquí, como sólo teníamos dos condiciones que eran exclusivas, se han traducido por un único IF. En cambio, las dos acciones, aprobar y suspender, están ambas traducidas. Observemos la instrucción GOTO de la línea 40. Su colocación es indispensable para separar las dos alternativas. En aquellos procesos con muchas condiciones, se emplearán gran cantidad de instrucciones IF y GOTO.

En general, cuando hay muchas instrucciones IF, éstas se encadenan de la siguiente manera. En el IF se coloca la condición preguntada de forma inversa. Por ejemplo, si la condición era «menor que tres» se escribe «mayor o igual que tres». Entonces cuando la pregunta del IF es cierta, significa que la condición inicial era falsa. Por tanto se pasa control a la siguiente condición, representado por una nueva instrucción IF. A su vez, esta instrucción pasará control al siguiente IF en caso de que no vuelva a cumplirse la condición. Este proceso se repite hasta que se cumpla una condición, es decir que el resultado del IF sea falso, o hasta agotar las condiciones. Entre dos instrucciones IF se colocan las instrucciones que corresponden al tratamiento a realizar cuando se cumple el IF previo. Por ejemplo, si en el programa anterior quisiéramos incluir las dos condiciones, el listado quedaría

```
10 INPUT N
20 IF N<5 THEN GOTO 50
30    PRINT "APROBADO"
40    GOTO 90
50 IF N<0  THEN GOTO 80
60    PRINT "SUSPENDIDO"
70    GOTO 90
80 PRINT "NOTA ERRONEA"
90 STOP
```

En la segunda condición se ha previsto que la nota fuera errónea (menor que cero). En las líneas 20 y 50 se observa la forma de encadenar las preguntas. Entre estas instrucciones IF se colocan las acciones que son escribir aprobado, suspendido o nota errónea. Es costumbre de los programadores situar las acciones cuya ejecución depende en conjunto de un IF, unos cuantos espacios en blanco hacia la derecha de modo que queda clara la dependencia. Aunque no aumenta la velocidad de cálculo y supone un pequeño gasto extra de memoria, esta costumbre ayuda bastante a hacer más legibles los programas.



7.6.4 Clasificación de piezas

Veremos a continuación un caso en el que utilizaremos dos condiciones independientes. Supongamos que en un taller mecánico se desea clasificar las piezas producidas en tres grupos, piezas de primera clase, de segunda clase y piezas que no cumplen las especificaciones. Para clasificarlas emplearemos dos características, el diámetro y la longitud. Si el diámetro está comprendido entre 99 y 100 son de primera clase. Si está entre 98 y 101 son de segunda clase. Fuera de estos márgenes, la pieza se rechaza. Con referencia a la longitud, la condición es más estricta. Si está comprendida entre 500 y 501, la pieza se acepta. En caso contrario se rechaza. En la figura 19 se muestra la tabla de decisión que refleja este conjunto de decisiones. La primera condición que se ha tenido en cuenta es si la longitud cumplía las normas. Fijémonos que si esta condición no se cumple, no hace falta analizar nada más, puesto que la pieza se desecha directamente.

En la tabla tenemos cuatro condiciones, de las cuales las tres últimas son dependientes (se refieren a la misma característica, el diámetro). La dependencia no es exclusiva, pues si la pieza tiene un diámetro comprendido entre 99 y 100, también es cierto que está comprendido entre 98 y 101. En cambio, la última condición es exclusiva puesto que si el diámetro cumple alguna de las dos condiciones anteriores, no puede ser cierto que esté fuera de márgenes. Finalmente la primera condición es independiente

Longitud entre 500 y 501	S	N	S	S
Diámetro entre 91 y 100	S	—	N	N
Diámetro entre 98 y 101	S	—	S	N
Diámetro menor de 98 o mayor 101	N	—	N	S
Clasificar como primera clase	X			
Clasificar como segunda clase			X	
Desechar		X		X

Figura 19: Tabla de decisión para evaluar las piezas producidas en un taller.

de las demás, pues la longitud puede ser correcta o incorrecta con independencia del diámetro.

Condiciones que no son significativas

La lista de tratamientos tiene tres acciones: clasificar como primera clase, clasificar como segunda clase o desechar. En la parte superior derecha de la tabla escribimos el conjunto de todas las combinaciones posibles de los estados de las condiciones. Observamos que si la primera condición es negativa, el resto de la columna tiene guiones. Esto se debe a que, seal cual sea el resultado de las demás condiciones, la acción a realizar es desechar la pieza. En general, se utilizan los guiones para indicar que, en aquel caso, la condición no es significativa.



7.6.5 Depuración de las tablas

Una vez se ha construido una tabla, se aplican una serie de criterios para determinar si la tabla es correcta o contiene errores de diseño.

Tabla completa sin redundancias ni contradicciones

En primer lugar debe comprobarse que la tabla es completa, es decir, que no se ha omitido ningún posible estado de las condiciones. A continuación, comprobaremos que la tabla no tiene redundancia ni contradicciones. Estos significa que no aparece dos o más veces el mismo estado de condiciones, con igual conjunto de tratamientos (redundancia) o con distinto tratamiento (contradicción).

Por otra parte, no han de existir condiciones indiferentes, es decir, que sea cual sea su estado, se efectúan los mismos tratamientos. Esta situación se aprecia si hay toda una fila en la entrada de condiciones que tiene únicamente guiones. Asimismo, no deben existir situaciones sin tratar. Esta situación tiene lugar cuando en los tratamientos existe una columna en blanco.

En resumen, podríamos decir que las ventajas de las tablas de decisiones son:

— Ofrecen una visión de conjunto del problema.

— Permiten una codificación exacta y rápida.

— Es fácil de comprobar que es completa y no redundante.

Como inconveniente está el hecho de que si hay muchas condiciones deja de ser operativa.

De todas maneras este último caso se trata de dividir las condiciones en grupos para que el número no aumente demasiado.

RESUMEN

En los casos prácticos se ponen de manifiesto el concepto de desarrollo jerárquico; un problema se desdobla en subproblemas más pequeños, con lo que resultan más fáciles de resolver.

Para resolver un problema es necesario hacer varios ordinogramas hasta alcanzar el nivel de detalle suficiente para poder traducirlo directamente a un lenguaje de programación.

Es frecuente encontrarse en programas donde después del diseño no se precisa ningún símbolo de final; es una buena costumbre buscar la condición para que el programa finalice y no tengamos que interrumpirles desde el exterior. No todos los programas se ejecutan sobre la pantalla.

En un ordinograma los puntos más importantes son las tomas de decisión, pues son las únicas que permiten bifurcar el flujo del programa.

La técnica del diseño jerárquico se aplica a todos los bloques, no únicamente a los de proceso.

En la fase de codificación es necesario realizar una lista de variables que contendrá: el nombre, una breve descripción y las restricciones en los valores que puede tomar.

En BASIC los símbolos de inicio y final no se traducen, la numeración de las líneas los contiene implícitamente.

Los procesos se traducen por instrucciones LET.

Los bloques de decisión se traducen por una instrucción IF y GOTO o con el IF...THEN...ELSE.

Los avances a saltos se hacen mediante la instrucción GOTO.

Las entradas se traducen por INPUT y las salidas por la instrucción PRINT.

Para los procesos de entrada y salida existen otros símbolos que aparte de indicar el tipo de proceso indican sobre qué dispositivo se realiza. Vea la figura 15.

El diseño de decisiones complicadas se realizan mediante una tabla de decisión. La tabla de decisión consta de cuatro partes que se disponen como:

Parte superior izquierda: Lista de condiciones.

Parte superior derecha: Estado de las condiciones.

Parte inferior izquierda: Lista de acciones.

Parte inferior derecha: Relación entre acciones y estados.

Se llama regla de decisión a una declaración que indica el conjunto de condiciones que se deben dar para ejecutar una acción.

El valor de una condición es el resultado de evaluar si una condición se cumple o no se cumple.

Cuando tenemos dos condiciones se pueden relacionar entre ellas según el esquema siguiente:

a) Independientes: El valor de una no depende de la otra.

b) Dependientes: El valor de una depende del valor de otra.

b.1) Exclusivas: Si una condición es cierta, la otra debe ser falsa.

b.2) Solapadas: El valor de una no fija el valor de la otra.

La traducción de las reglas de decisión a un lenguaje de programación se hace mediante el caso de alternativas múltiples. Se en-

cadenan los IF que niegan las condiciones, de tal manera que el bloque a realizar para la condición queda debajo del IF.

En las tablas de decisión, un mismo tratamiento se puede desencadenar por la acción de distintos estados de condiciones.

Cuando el resultado de un estado de condiciones es el mismo, tanto si una condición se cumple como si no se cumple, se dice que la condición no es significativa en este estado. O el estado es indiferente a esta condición.

Se debe comprobar que la tabla de decisión es completa, es decir, contiene todos los estados de las condiciones.

No tiene que haber redundancias ni contradicciones. Esto ocurre cuando hay por lo menos un estado dos veces. Cuando los tratamientos son iguales es redundancia, en cambio si el tratamiento es distinto es contradicción.

No debe haber ningún estado con las condiciones no significativas.

Debe haber un tratamiento y uno solo para cada estado de condiciones.

EJERCICIOS DE AUTOCOMPROBACION

Complete las frases siguientes:

21. La resolución de un problema con ordenador requiere generalmente varios ordinogramas con diferente nivel de ....

22. Es necesario tener en cuenta la condición de .... de un programa.

23. Los bloques de decisión también se realizan con un diseño ....

24. La primera fase de codificación consiste en hacer una .... de variables.

25. La segunda fase de la codificación consiste en traducir los .... del ordinograma a .... del lenguaje de programación.

26. Para los procesos de entrada y salida existen símbolos especiales que además de indicar el proceso indican el ....

27. Las tablas de .... son una ayuda en el diseño de tomas de decisión complicadas.

28. A la declaración del conjunto de condiciones que se deben cumplir para realizar una acción se le denomina .... de decisión.

29. El valor de una condición es evaluar si una condición se .... o no se cumple para un caso determinado.

30. Las .... múltiples es la mejor manera de traducir la tabla de decisión a un lenguaje de programación.

31. Distintos estados de condiciones pueden generar el mismo ....

32. Una tabla de decisión es .... cuando se consideran todos los estados posibles y sólo los estados posibles.

33. Una tabla de decisión tiene redundancia cuando se considera un estado dos veces y se le aplica el .... tratamiento.

34. Una tabla de decisión es contradictoria cuando se considera un estado dos veces y se le aplica .... tratamiento.

35. En cada estado de condiciones debe haber una .... por lo menos que sea significativa.

Encierre en un círculo la letra que corresponda a la alternativa correcta.

36. Los símbolos de toma de decisión en los ordinogramas se traducen en BASIC como:

    a) LET
    b) IF... THEN y GOTO
    c) GOTO
    d) INPUT

37. Los símbolos de proceso de un organigrama se traducen en lenguaje BASIC como:

    a) LET
    b) IF...THEN y GOTO
    c) GOTO
    d) INPUT

38. Las flechas que producen saltos en un ordinograma se traducen en lenguaje BASIC como:

    a) LET
    b) IF...THEN y GOTO
    c) GOTO
    d) INPUT

39. Los procesos de entrada de un ordinograma se traducen en lenguaje BASIC como:

    a) LET
    b) IF...THEN y GOTO
    c) GOTO
    d) INPUT

40. El símbolo de un trozo de papel cortado en un ordinograma significa una salida por:

    a) Pantalla.
    b) Disco magnético.
    c) Cinta magnética.
    d) Impresora.

41. Las condiciones ser del signo del zodíaco acuario y ser del signo de tauro son

    a) Independientes.
    b) Dependientes exclusivas.
    c) Dependientes solapadas.
    d) No es pregunta binaria.

42. Las condiciones de ser rico y poseer un coche deportivo son

a) Independientes.
b) Dependientes exclusivas.
c) Dependientes solapadas.
d) No es pregunta binaria.

43. Las condiciones la mesa es blanca y la mesa tiene cuatro patas son:

a) Independientes.
b) Dependientes exclusivas.
c) Dependientes solapadas.
d) No es pregunta binaria.

44. Las condiciones el lugar está situado al Norte y el lugar está situado el Oeste de aquí son

a) Independientes.
b) Dependientes exclusivas.
c) Dependientes solapadas.
d) No es pregunta binaria.



Capítulo 8
• El manejo del cassette.

ESQUEMA DE CONTENIDO

8.0 OBJETIVOS

El objetivo de este capítulo es enseñarle a manejar dos dispositivos adicionales del ordenador personal, el cassette y la impresora.

El cassette es el medio para hacer sobrevivir los programas a la desconexión del ordenador a la corriente. La cinta magnética del cassette conserva memorizado el programa, de la misma forma que lo hace con sus canciones preferidas.

Es muy importante que ponga atención en la manipulación del cassette, pues es frecuente que tenga problemas en el mecanismo de grabación y recuperación. La causa es que la grabación debe estar hecha, para que la entienda el ordenador; si el ordenador no entiende la «música» grabada en el cassette es como si el programa hubiera desaparecido.

Cuando usted graba una canción, los defectos se notan en la reproducción, y en caso extremo puede resultarle una canción desagradable. Normalmente usted buscará un cassette de alta fidelidad para arreglar el problema.

Le hemos de advertir desde el inicio que éste no es el camino en el caso del ordenador. Los cassettes de alta fidelidad son adecuados para el oído humano, pero son muy inadecuados para el oído del ordenador.

Le insistimos en que no malgaste su dinero comprando mejores cassettes. Si el que tiene no funciona, pregunte en la tienda por cassettes preparados para el ordenador; son baratos; los reconocerá porque aparecen, en la rotulación de sus teclas, las palabras SAVE, LOAD.

Una vez resueltas estas dificultades iniciales, el cassette le servirá para guardar sus programas y poderlos recuperar. También podrá intercambiar programas con sus amigos si disponen del mismo tipo de ordenador.

Verá que hay instrucciones para verificar que el contenido del cassette coincide con el de la memoria. Utilícela aunque le lleve un poco más de tiempo, es muy desagradable comprobar que el contenido de la cinta no lo puede escuchar el ordenador una vez hemos cambiado de programa en la memoria.

La impresora es un complemento muy interesante para el ordenador pues permite dejar los programas y resultados escritos de forma permanente. La pantalla es más rápida pero tiene dos inconvenientes: en primer lugar las cosas desaparecen, y por otra parte su contenido es muy limitado.

Aunque no disponga de impresora, estudie las instrucciones con atención, son realmente muy parecidas a las de la pantalla; verá que no presentan ninguna dificultad especial respecto a las que ya conoce.

También en este capítulo aprenderá la organización básica de la memoria cuando ésta contiene el programa BASIC; recuerde que aunque hablemos del lenguaje BASIC no se trata más que de un programa que está colocado dentro del ordenador. Este programa consume a su vez memoria para su organización, y nuestro programa no puede ocupar totalmente la memoria de que dispone el ordenador.

8.1 ALMACENAMIENTO DE LOS PROGRAMAS

Las instrucciones que hemos aprendido hasta ahora, ya nos permiten la realización de programas complejos.

En muchos casos, estos programas pueden resultar largos de escribir. Por otra parte, son programas que nos pueden resultar útiles para ejecutarlos en más de una ocasión.

Sabemos que, cuando escribimos un programa, el ordenador nos lo guarda en la memoria ordenado por número de línea, permitiendo que lo visualicemos —escribiendo LIST—, que lo ejecutemos —escribiendo RUN—, o que lo modifiquemos, ya sea escribiendo nuevas instrucciones o bien borrando o cambiando alguna de las ya existentes.

Sin embargo, nos encontramos con un problema, la memoria del ordenador no es permanente. En el momento que lo apaguemos perdemos todo el programa.

Dado que no podemos mantener permanentemete conectado el ordenador, y que además nos interesa poder guardar programas distintos, se hace necesaria una forma de almacenar los programas de manera permanente, que nos permita recuperarlos en cualquier momento.

Una forma simple de hacerlo, es utilizando un cassette.

Sabemos que un cassette nos permite grabar sonido, almacenándolo sobre un soporte magnético. Una vez grabado el sonido, podemos reproducirlo siempre que lo deseemos. Esta forma de grabación tiene además otra ventaja: puede borrarse y grabarse de nuevo tantas veces como se desee.

El programa, que tenemos almacenado en la memoria del ordenador, puede almacenarse también sobre un soporte magnético, utilizando una grabadora de cassette, con las mismas posibilidades y ventajas que si se realiza una grabación de música o sonido.

No es necesario tener un tipo especial de grabadora. De hecho, se pueden utilizar la mayoría de grabadoras existentes en el mercado. Aun más, es mejor de cara al ordenador, utilizar una grabadora portátil sencilla antes que una grabadora estéreo muy sofisticada, que puede plantear más problemas, debido a su mayor complejidad, de cara a la grabación. Tenga en cuenta que si utiliza una grabadora con control de los tonos graves y agudos, y estos no están bien regulados, puede verse afectada la fidelidad de la grabación. En este caso pueden perderse impulsos y el resultado sería que las palabras del programa no quedarían bien registradas.

Una grabadora de alta calidad está bien adaptada al oído humano pero no al «oído» del ordenador

En realidad una grabadora que no sea de alta fidelidad suele funcionar mejor que una de alta fidelidad. Esta última está preparada para que los sonidos sean lo más agradable posible al oído humano; es decir, adaptan el sonido a las características del oído. Como verá más adelante, el sonido que debe guardar el ordenador no es nada agradable al oído humano; por lo tanto, toda adaptación de este sonido al oído humano lo que hace es deformarlo, con la repercusión de que entonces el ordenador no se entera de lo que está grabado.

Por otra parte, resultará muy útil que la grabadora disponga de un contador de vueltas. Esto nos servirá para conocer la organización de la cinta. Del mismo modo que usted puede conocer en que vuelta empieza cada una de las canciones grabadas en una cinta de música, le resultará útil saber la lista de programas y la situación (número de vuelta) en que se ha grabado cada uno.

La grabadora debe tener una conexión de entrada para micrófono y una conexión de salida para auriculares.

Los requisitos de la conexión

El ordenador ya está preparado para realizar la conexión con el cassette. Este conexión se realiza en principio mediante dos cables: un cable debe conectar la entrada del micrófono en la grabadora con el correspondiente conector de su ordenador, y otro cable debe conectar el enchufe de salida de los auriculares con el correspondiente en el ordenador.

En el capítulo de prácticas con el microordenador, usted encontrará la explicación detallada para conectar su modelo particular de ordenador con la grabadora.

El programa no se almacena en la memoria del ordenador tal como lo vemos escrito, sino que, como ya sabemos, lo que se almacena en realidad es el equivalente binario (compuesto exclusivamente de ceros y unos), de cada uno de los caracteres. Esto es lo que se registra en la cinta magnética cuando se graba el programa. Por otra parte, cuando se lee un programa que se ha grabado previamente, se pasa del cassette a la memoria del ordenador, y una vez cargado en ella, se podrá ya visualizar, ejecutar o modificar, tal como se desee, de la misma forma que si hubiéramos escrito el programa mediante el teclado.

La cinta magnética es pues, un tipo de memoria permanente; los programas pueden ser almacenados en ella, y podrán ser leídos tantas veces como sea necesario —de la misma forma que se puede escuchar una cassette de música las veces que se quiera— mientras el usuario no decida borrarlos.

De esta forma pues, los programas se mantienen almacenados independientemente de que se apague o no el ordenador. Debe tener en cuenta, sin embargo, que, cuando se graba un programa, se transfiere a la cinta el contenido exacto de la memoria actual de la máquina. Por tanto si usted realiza modificaciones en el programa y no lo vuelve a grabar, en el momento que usted apague la máquina, habrá perdido todo lo que haya modificado.

Hay que guardar el programa definitivo

Por otra parte, las cintas magnéticas dan transportabilidad a nuestros programas: nos permiten intercambiar programas de una a otra máquina, siempre que éstas sean del mismo modelo.

En este aspecto, pueden producirse sin embargo, algunas dificultades, dependiendo de las condiciones en que haya sido grabada la cinta.

Veamos ahora cuáles son los comandos que nos permiten la grabación y lectura de los programas, así como la forma de utilizar cada uno de ellos.



8.2 LA ORDEN DE LECTURA

LOAD

Para leer un programa de una cinta de cassette y cargarlo en la memoria del ordenador disponemos de dos comandos.

El primer comando produce, tal como hemos dicho, el paso del programa del cassette a la memoria, pero borrando previamente todo el contenido de ésta.

Así, haya lo que haya en la memoria, en el momento de ejecutar este comando se borrará automáticamente, y será sustituido por el nuevo programa que se ha ordenado leer. Para ejecutar este comando se escribe la palabra

LOAD

Y a continuación el nombre, entre comillas, del programa que se desea cargar en memoria.

Así, por ejemplo, si deseamos cargar un programa que tiene de nombre «A» (la manera dar un nombe y saber que está en la cinta lo veremos más adelante), deberemos escribir:

```
LOAD "A"
```

Para cargar es necesario apretar la tecla play de la grabadora

Observaremos que este comando tiene varios efectos. El efecto evidente es que empieza a girar la cinta del cassette (para que esto ocurra es necesario que la tecla de sonar (play) de la grabadora esté apretada), mientras el ordenador localiza el programa llamado «A». Naturalmente, el usuario deberá colocar la cinta que contiene el programa en la posición adecuada, de forma que el punto en el que se inicia la búsqueda esté situado antes del punto en que se encuentra el programa, considerando el sentido de avance de la cinta.

En este aspecto es práctico lo que habíamos dicho en cuanto a saber en qué vuelta empieza cada programa, ya que podremos situar la cinta de forma que el cabezal de lectura del cassette esté muy cerca del inicio del programa.

Supongamos que habíamos grabado un programa a partir de la vuelta número 10. El proceso de carga, supuesto que tenemos cassette con contador de vueltas, será el siguiente. En primer lugar colocaremos la cinta —atendiendo a situarla en la cara que corresponda— y la rebobinaremos completamente. Situaremos ahora el contador de vueltas a cero y haremos avanzar la cinta hasta la vuelta 8 o 9. En este punto situaremos la grabadora de forma que el ordenador tenga el control de la misma. Este será el momento de ejecutar el comando LOAD.

Preparación de la grabadora

Naturalmente, podríamos simplemente haber rebobinado la cinta, dejando que el propio ordenador buscara el programa desde el principio. Sin embargo este proceso sería más lento, pues la máquina para buscar el principio del programa ha de ir leyendo el contenido de la cinta, mientras que nosotros manualmente podemos hacerla avanzar a mayor velocidad.

De todas formas, si no se dispone de contador de vueltas o no se conoce la organización de la cinta, no quedará más remedio que realizar la búsqueda de forma más lenta, a base de que el ordenador la vaya leyendo toda desde el principio.

En cualquiera de los dos casos, una vez situada la cinta en el lugar adecuado, para cargar el programa que habíamos grabado en la vuelta 10 de la cinta con la etiqueta «A», escribiremos:

```
LOAD "A"
```

Pondremos en marcha el cassette, es decir apretaremos la tecla de sonar (play). Inmediatamente veremos como empieza a girar la cinta. En el momento que lleguemos a la vuelta 10, se empezará a cargar el programa.

Generalmente el ordenador da un mensaje por pantalla indicando que está realizando la lectura del programa. Este mensaje puede diferir dependiendo de su modelo particular de ordenador.

Cuando se haya cargado todo, se detiene la cinta o deberemos detenerla según veremos en el capítulo de prácticas, y a partir de este momento ya podrá trabajar con el programa. Es aconsejable antes de ejecutarlo, cerciorarse de que ha sido cargado correctamente. Esto generalmente, se puede ver haciendo el listado del programa por pantalla.

Se debe computar si el programa se ha leído correctamente

Si se observan diferencias con lo que esperaba (desorden en los números de línea, instrucciones incorrectas, símbolos extraños, etc.) es que la carga no se ha realizado correctamente. En este punto no intente corregir los errores, ya que éstos, aunque le parezcan pequeños, indican la no coincidencia del programa en memoria y en cinta. En estos casos se pueden producir incoherencias, tales como borrar una línea que aparece en el listado y no poder hacerlo aunque se dé la indicación para ello.

Esto se produce porque el programa cargado se encuentra en desorden o mal acondicionado. Piense que lo que usted ve no es más que la traducción del código interno de la máquina (ya sabe que éste está compuesto únicamente de informaciones binarias —ceros y unos— y todos y cada uno de ellos se han de leer correctamente para que el programa funcione).

Es pues mejor empezar de nuevo, controlando en primer lugar las condiciones del sistema (volumen adecuado del cassette, situación correcta de las conexiones, etc.) rebobinando la cinta hasta el lugar adecuado y ejecutando una vez más el comando de carga (LOAD).

Al cargar un programa, el volumen del cassette es muy importante. Si se observan dificultades, de forma que el ordenador no lo encuentra aunque realicemos todas las operaciones adecuadas, con la cinta en la posición correcta y las conexiones bien puestas, intente modificar el volumen del cassette y repita el proceso de carga.

Este factor es determinante para que la máquina pueda «oír» correctamente el programa. Por tanto, si el volumen no tiene el valor adecuado, podemos tener problemas de lectura. Controle pues el volumen hasta ajustarlo correctamente; generalmente resulta adecuado situarlo en el valor medio entre el máximo y el mínimo, y sobre este punto irlo ajustando, realizando pequeñas variaciones.



8.3 LA ORDEN DE GRABAR

SAVE

Una vez conectada de forma correcta la grabadora, para que la grabación se realice, deberemos dar la orden desde el teclado.

La orden de grabación consta de dos partes. En primer lugar, se escribe la palabra

SAVE

y a continuación el nombre o etiqueta del programa que se desea grabar.

La etiqueta del programa es un texto, y como tal debe escribirse entre comillas. Por otra parte, debe empezar siempre con una letra.

Un ejemplo de este comando sería:

```
SAVE "A"
```

La palabra SAVE significa «salvar» en inglés, y corresponde a la idea de conservar, almacenar o guardar.

La palabra que se escribe a continuación no forma parte del programa, se utiliza únicamente como una etiqueta, tal como hemos dicho. Esta etiqueta da nombre al programa, y nos servirá para identificarlo en el momento en que se desee recuperar.

En principio se puede utilizar un nombre cualquiera, que puede escoger el usuario, teniendo únicamente presente escribirlo entre comillas, y que empiece con una letra del abecedario. Generalmente se escoge una palabra que haga referencia a la utilidad del programa. Sin embargo, la longitud de esta palabra es variable, el número de letras que se pueden utilizar dependerá del modelo de ordenador. Para conocer con exactitud este número, deberá consultar el correspondiente capítulo de prácticas con el microordenador.

En algunos tipos de máquinas pueden utilizarse el comando SAVE para almacenar variables, de forma que se puede conservar el valor de las mismas guardado en la cinta. Insistimos en que esto sólo es posible en algunos modelos de ordenador, por tanto, deberá consultar el capítulo de prácticas del microordenador, para saber si ésta lo permite. Y en caso afirmativo, conocer cuál es la sintaxis correcta para ello.

La variante del SAVE para carga y ejecución de un programa

Otra variante del comando SAVE, que poseen algunas máquinas, es la utilización del comando incluyendo un número de línea. En este caso se escribe el comando y el nombre del programa, tal como ya hemos visto, pero a continuación se escribe la palabra LINE, seguida de un número de línea. Este número debe ser un número de línea existente en el programa.

El efecto del comando SAVE escrito de esta forma será producir la ejecución del programa empezando por el número de línea indicado, cuando se lea en la cinta el programa que haya sido grabado utilizando esta modalidad del comando SAVE. La sintaxis es pues en este caso:

```
SAVE "nombre" LINE numero
```

Así por ejemplo, puede escribirse:

```
SAVE "A" LINE 10
```

Esto hará que en el momento que recuperemos el programa grabado de esta forma, éste empiece a ejecutarse empezando por la línea 10. Más adelante veremos cuáles son las diversas posibilidades que ofrece esta modalidad del comando SAVE. En todo caso asegúrense consultando la lec-

ción de prácticas con el microordenador de que su ordenador ofrece esta posibilidad, y en este caso, cual es la sintaxis exacta.

Ha llegado ahora el momento de probar el funcionamiento de este comando. Para ello deberemos en primer lugar realizar las oportunas conexiones de ordenador con la grabadora, siguiendo las indicaciones que daremos en el capítulo de prácticas con el microordenador.

Preparación de la grabadora para guardar los programas

El siguiente paso es disponer de una cinta de cassette. Esta cinta no tiene por qué ser de un modelo especial; también en este caso es más recomendable el uso de cintas corrientes antes que de cintas de alta calidad, ya que éstas no suponen mejora —en algunos casos incluso complican— las condiciones de grabación. Antes de utilizar una cinta para grabar un programa, asegúrese de que dispone de un trozo de cinta limpio suficiente para realizar la grabación, ya que es conveniente dejar un par de vueltas de cinta vacías antes de empezar a grabar el programa. La longitud total de cinta necesaria depende evidentemente de la longitud del programa a grabar. Tenga cuidado al grabar un programa en una cinta que contiene otros programas grabados previamente, ya que si el programa ocupa más espacio de cinta del que queda vacío entre los otros programas grabados, se empezará a grabar sobre el programa ya existente, con lo que éste quedará destruido y no se podrá utilizar.

Para cerciorarse de que no hay nada grabado en un trozo de cinta, puede utilizar el cassette de la misma forma que lo haría para oír música, desconectándolo del ordenador, y «escuchando» la cinta. Si existe algún programa grabado, oíremos un sonido (una especie de pitido) característico, que empezará a oírse en el punto donde el programa ha sido grabado.

Supongamos ahora que tenemos una cinta donde no hay nada grabado. La colocaremos en el cassette y dejaremos pasar un par de vueltas. A continuación procederemos a grabar el programa que tenemos en memoria, que puede ser cualquiera, y que llamaremos programa «A». Escribiremos pues:

```
SAVE "A"
```

La grabadora debe tener apretadas las teclas play y rec para realizar el comando SAVE

La grabadora debe estar preparada para grabar, es decir, deberemos apretar las teclas «play» y «rec» simultáneamente. En algunos modelos de ordenador esto se puede hacer mucho antes de dar la orden de SAVE pues el ordenador pone en marcha la cinta mediante el control remoto de la misma. En otros, en cambio, deberemos apretar las teclas después de haber dado el comando a la máquina y bajo la indicación de la misma.

Veremos que empieza a girar la cinta mientras se realiza la grabación, y según el modelo de ordenador, aparece o no el mensaje informativo de que está grabando. Del mismo modo, cuando se ha grabado todo el programa, veremos que la cinta se detiene en los modelos de ordenadores que tienen control remoto, o, deberemos detenerla nosotros mismos; esta detención es importante; si hay grabados programas posteriormente puede borrarlos si nos olvidamos.

También en función del modelo nos da un mensaje indicando el fin del proceso, o simplemente indica que está preparado para recibir nuevas órdenes, y nos devuelve el control.

En este punto, tendremos ya registrado el programa de forma permanente. Sin embargo, para tener la seguridad de que la grabación se ha realizado correctamente, hay que comprobar que coincida lo grabado con el programa en memoria. Para ello existe un comando especial, que veremos en el siguiente apartado.

Resultará conveniente además grabar el programa más de una vez, evidentemente en partes distintas de la cinta, para asegurar la posterior recuperación del mismo; si una falla, tendremos la otra grabación. Es muy difícil que las dos grabaciones sean erróneas.

8.4 LA ORDEN DE VERIFICAR

La grabación de un programa es en principio un proceso muy sencillo, basta escribir la orden de grabación y esperar a que ésta se realice.

Sin embargo, dados los distintos factores que intervienen en él —ordenador, grabadora, cinta magnética, conexiones...— pueden producirse problemas en el momento de grabar, de manera que aunque la máquina indica que ha realizado la grabación (detiene la cinta y devuelve el control), en realidad pueden encontrarse discrepancias entre el programa contenido en la memoria del ordenador, y el programa que ha sido grabado. Estas discrepancias, si se producen, pueden llegar a impedir —de hecho es lo que suele suceder en la mayoría de los casos— la posterior recuperación del programa.

El proceso de verificación es una necesidad

Para evitar encontrarse con sorpresas (perder un programa, especialmente si es largo, resulta bastante desagradable) es conveniente siempre realizar la verificación de lo grabado inmediatamente después de finalizar la grabación.

Existe para ello un comando específico. El nombre de este comando puede diferir según el modelo de máquina con el que trabajemos. La forma más corriente de escribirlo es utilizando la palabra:

VERIFY

Y a continuación el nombre, entre comillas, del programa que acabamos de grabar. La palabra VERIFY significa verificar o comprobar, en inglés.

En principio, la mecánica del comando es la misma en cualquier máquina, aunque pueda diferir la sintaxis. Sin embargo, en el anexo correspondiente, usted encontrará la explicación detallada para escribir correctamente este comando en su modelo de ordenador. Por lo demás, insistimos en que la idea de funcionamiento es la misma para todas las máquinas.

Así, escribiremos por ejemplo:

```
VERIFY "A"
```

para comprobar la correcta grabación del programa que acabamos de grabar y hemos etiquetado como programa «A».

El efecto que desencadena la ejecución del comando de verificación

es la comparación del programa contenido en memoria con el programa contenido en la cinta magnética.

Dado, pues, que se ha de leer lo que hemos grabado, antes de realizar la verificación, deberemos rebobinar la cinta hasta un poco antes del punto en que hemos iniciado la grabación.

Si se detecta que hay diferencias entre el programa grabado y el programa existente en memoria, la máquina dará el correspondiente mensaje, indicando que han habido errores en la grabación. El tipo de mensaje depende una vez más del modelo de ordenador, y lo encontrará explicado en el correspondiente capítulo de prácticas.

El mecanismo es parecido al LOAD pero sin modificar la memoria del ordenador

Si la grabación ha sido correcta —no hay discrepancias— algunas máquinas lo indican dando un mensaje y otras devuelven simplemente el control al usuario sin indicar nada más. Es el momento de detener el movimiento de la cinta. En caso contrario habrá que repetir de nuevo la grabación, asegurándose de que tiene el sistema bien conectado, de que en la cinta no haya nada grabado (al menos en la zona en que se inicia la grabación del programa) y de que el cassette está ajustado correctamete. Una vez grabado deberá verificar de nuevo, repitiendo el proceso hasta que el resultado de la verificación sea el correcto.

Este comando puede utilizarse también para verificar datos grabados, en los modelos de ordenador que lo permiten. Para conocer en este caso la sintaxis de comando, consulte el capítulo de prácticas.



8.5 OTRA ORDEN DE LECTURA: MEZCLAR PROGRAMAS

Hemos visto ya los dos comandos que nos permiten almacenar un programa de forma permanente en una cinta magnética. Vamos a ver ahora los nuevos comandos que nos servirán para leer —cargar en la memoria del ordenador— los programas que se hayan grabado en una cinta de cassette.

El comando LOAD nos permite, tal como se ha visto, cargar en la memoria del ordenador, un programa grabado en una cinta magnética.

Hemos dicho también, que al utilizar este comando para leer un programa, se pierde el contenido de la memoria, siendo sustituido por el nuevo programa que haya sido cargado.

Existe sin embargo otro comando para leer programas, cuyos efectos son los mismos que el comando LOAD en el aspecto de lectura y control del cassette, pero que difiere de aquél en que no se borra la memoria antes de cargar el nuevo programa. De esta manera se consigue un efecto muy interesante, que es la posibilidad de mezclar programas. Para ejecutar este comando, se escribe la palabra

MERGE

Y a continuación el nombre entre comillas del nuevo programa que se desea cargar. La palabra MERGE significa mezclar, en inglés.

Así, supongamos que tenemos en memoria un programa cualquiera, que llamaremos programa A. Si deseamos cargar otro programa, por ejemplo el programa B, deberemos escribir

```
MERGE "B"
```

La posición de la grabadora es la misma que en el caso del comando LOAD

¿Cuál es el efecto de este comando?

Tal como hemos dicho, en cuanto a la lectura del cassette se comporta exactamente igual que el comando LOAD, por tanto, deberemos situar la cinta de forma adecuada, realizar las conexiones para la grabación, y ajustar correctamente el volumen del cassette.

Sin embargo, veamos cual es su efecto en cuanto a la memoria del ordenador. Hemos dicho que no borra el programa que ya teníamos en ella. Pero esto sólo es verdad hasta cierto punto, pues los dos programas nos quedarán mezclados, dependiendo de los números de línea de cada uno.

El efecto del programa cargado con el comando MERGE de cara al resultado de la mezcla de éste con el contenido en memoria, sería el mismo que si lo introducimos a través del teclado.

Veamos una a una las distintas situaciones que se pueden dar:

1. Los valores de los números de líneas del nuevo programa son todos superiores a los del programa ya existente en memoria. En este caso, el nuevo programa se situará a continuación del anterior.

2. Los valores de los números de líneas del nuevo programa son todos inferiores a los del programa ya existente en memoria. En este caso el nuevo programa se situará antes que el anterior.

3. Existen líneas en ambos programas con el mismo número. En este caso, las líneas del nuevo programa sustituyen a las del programa anterior que lleven el mismo número.

4. Los números de líneas de uno y otro programa están intercalados. En este caso se mezclan los dos programas, de forma que las instrucciones se colocan estrictamente en orden creciente de número de línea, independientemente del programa al que pertenezcan.

Veámoslo con un ejemplo. Supongamos que escribimos y grabamos en la cinta el siguiente programa, que llamaremos programa «B»:

```
30   LET A=0
50   IF A>10 THEN END
60    PRINT A*5
70    LET A = A+1
80    GOTO 50
```

Este programa nos escribirá el resultado de multiplicar el 5 por los valores comprendidos entre 0 y 10, ambos inclusive, tal como comprobará al ejecutarlo. Grábelo y verifique la grabación.

Ahora borramos de la memoria el programa que acabamos de pasar a la cinta, escribiendo NEW, y escribimos un nuevo programa, que llamaremos programa A:

```
10  INPUT A
20  INPUT B
30   PRINT A*B
40   PRINT B/A
```

Tal como está escrito, este programa nos preguntará dos números y nos escribirá el valor del producto de ambos y de la división del segundo por el primero.

Ahora utilizaremos el comando MERGE para cargar el programa que habíamos salvado previamente con el nombre de programa B. Después de conectar el cassette y colocar la cinta en posición, escribiremos:

```
MERGE "B"
```

Al finalizar la ejecución del comando MERGE, el programa resultante será:

```
10  INPUT A
20  INPUT B
30  LET A = 0
40  PRINT B/A
50  IF A>10 THEN END
60       PRINT A*5
70       LET A = A+1
80       GOTO 50
```

Veamos el porqué de este resultado. En primer lugar, vemos que la línea 30 del programa «A» ha sido sustituida por el contenido de la línea 30 del programa «B». Esto es, porque tal como hemos dicho, había dos líneas con igual número en ambos programas, y la del último que cargamos ha sustituido a la del que teníamos en memoria, de la misma forma que si nosotros, teniendo únicamente el programa A en memoria, hubiéramos escrito:

```
30    LET A = 0
```

Esta línea sustituiría a la que existía con el mismo número, tal como ha sucedido al hacer el MERGE del programa B.

Por otra parte, vemos que la línea 40 del programa A aparece entre la 30 y la 50 del B, intercalándose tal como hemos dicho de forma que se matenga el orden de números de línea. El resto de las líneas ocupan también la posición que les corresponde según su número.

Veamos ahora lo que sucede si ejecutamos el programa.

Al escribir RUN, y después de dar los dos valores que nos pide la máquina, observamos que se produce un error. Si observamos con atención el listado resultante de la mezcla, veremos la causa del mismo. En la línea 30 el valor de A, independientemente del que se haya escrito previamente, se iguala a cero. Dado que a continuación —en la línea 40— se efectúa la división de un número cualquiera —que habremos introducido— por el valor de A, obtendremos un error, dado que la división de cualquier número por cero no tiene sentido para la máquina (El resultado de esta división es «infinito» desde el punto de vista matemático, pero «infinito» es un concepto, no un valor; por tanto, la máquina no puede utilizarlo y da un error).

Al mezclar programas las líneas del programa que se carga sustituyen a las que tenemos en memoria

El problema ha surgido, pues, en el momento de mezclar los dos programas, pues cabe suponer que se habían probado por separado, dando resultados correctos. En este caso, el problema puede parecer muy simple, dado que los dos programas son cortos, y el origen del error se ve fácilmente. Sin embargo, en programas largos y complejos debe prestarse mucha atención en el momento de realizar mezclas, pues podemos encontrarnos con sorpresas, tal como en este ejemplo se ha pretendido poner de manifiesto.

Además de poner atención a la existencia de números de línea iguales o intercalados, cosa que resulta fácilmente controlable, debe atenderse a la posibilidad de utilizar los mismos nombres para las variables en ambos programas. Esto puede ser importante, pues nosotros podemos hacer suposiciones sobre el valor de las mismas que no sean reales, pues puede existir alguna instrucción que las modifique sin nosotros advertirlo, lo cual puede llevarnos a resultados erróneos.

En este sentido, si se utilizan las mismas variables en ambos programas debemos asegurarnos de que cada vez se inicializan al valor correcto.

Al mezclar programas hay que ser cuidadoso con el nombre de las variables

Supongamos que, para que no nos interfiera tal como hemos visto que sucedía borramos la línea 30 del programa B antes de salvarlo, comprobamos que funciona correctamente, y volvemos a realizar el MERGE con el programa A, sin haber modificado éste.

El resultado será en este caso:

```
10    INPUT A
20    INPUT B
30    PRINT A*B
40    PRINT B/A
50    IF A>10 THEN END
60          PRINT A*5
70          LET A = A+1
80          GOTO 50
```

Ahora no nos dará error al ejecutar el programa. Comprobémoslo escribiendo RUN, e introduzcamos dos valores cualesquiera, por ejemplo 123 y 35. Obtendremos el producto de ambos y la división del segundo por el primero, como ya esperábamos, pero en este caso no obtendremos la tabla del cinco.

Esto se debe naturalmente, a que la variable A, que se utiliza en el primer programa, es en el segundo la variable de control de un bucle. Dado que no lo inicializamos antes de empezar el bucle, en la línea 50 no se cumple la condición de que A sea inferior a 10 —en efecto su valor es 125, tal como nosotros mismos hemos escrito—, por tanto el programa acaba sin realizar nuestro propósito.

Al tenerlo por separado podríamos haberlo ejecutado sin inicializar esta variable y sin tener problemas porque cada vez que se ejecuta un programa todas sus variables valen cero, mientras no sean modificadas por el mismo. Sin embargo, esto no es siempre cierto, hay ordenadores en los cuales hay que definir previamente los valores iniciales de las variables. Al tener los dos encadenados A toma un valor distinto de cero. La segunda parte del programa se realizará parcialmente o no se realizará en función de este valor.

De todo lo dicho pues, se concluye que es muy importante controlar líneas y variables antes de mezclar dos programas, y en todos los casos resulta aconsejable inicializar las variables que se vayan a utilizar, aunque parezca que puedan tomar el valor adecuado sin que se lo asigne el propio programa.

Por último, debe tenerse en cuenta que no todas las máquinas —aunque si la mayoría—, disponen de este comando. Si la suya en particular no le ofrece esta posibilidad, no le quedará más remedio que cargar el programa más largo con el comando LOAD y teclear las instrucciones del programa que desee unir al anterior.



8.6 ENCADENADO DE PROGRAMAS

Si utilizamos el comando LOAD para cargar un programa que había sido salvado con la indicación de línea (recuerde el comando

SAVE «nombre» LINE número

que hemos explicado anteriormente), observará que inmediatamente después de ser cargado, empieza la ejecución del mismo, sin necesidad de dar indicación alguna.

Esta posibilidad ofrece gran interés, aunque no lo parezca en principio, dado que nos permite encadenar programas.

Veamos en qué consiste y qué ventajas aporta la posibilidad de encadenar programas.

Hasta ahora hemos visto programas relativamente cortos y no demasiado complicados, que nos permiten realizar con ventaja tareas diversas. Pero, ¿qué sucede cuando necesitamos programas complejos, que deben ralizar varias tareas? Cada una de estas tareas puede ser larga, de forma que llega un momento en que el programa va creciendo, hasta poder llegar al límite de la capacidad de memoria de nuestra máquina. Y, aunque no lleguemos a este límite, los programas largos son más difícilmente controlables, y traen más complicaciones a la hora de manipularlos.

Encadenar programas permite separar tareas con

La posibilidad de encadenar programas nos permitirá resolver este problema, ya que permite la separación de cada tarea, de forma que cada

más claridad y menor complejidad

programa que realiza un trabajo concreto se puede tener por separado, y una vez ha terminado su cometido, se carga el siguiente programa —sin necesidad de que el usuario intervenga— pasando a realizar la siguiente tarea prácticamente de una forma transparente para aquél.

Veamos cuál es el proceso a seguir.

En primer lugar, salvaremos los programas que se deban encadenar, uno a uno, con el comando:

```
SAVE "nombre" LINE numero
```

manteniendo el mismo orden en que posteriormente se deban ejecutar.

Al final de cada uno de estos programas se deberá incluir una instrucción especial. Esta instrucción será la orden de cargar el siguiente programa, que se escribirá numerada.

De esta forma, para iniciar el proceso únicamente se deberá dar la orden para cargar el primer programa. Este se ejecutará directamente, y cuando haya finalizado su trabajo, realizará la carga del siguiente programa a ejecutar. Si a este le sigue otro, se repetirá el proceso, y así tantas veces como se desee.

Naturalmente, cada nuevo programa a cargar debe encontrarse en la cinta, a continuación del que esté en memoria en cada momento. En principio pues, podremos encadenar sin necesidad de realizar manipulación ninguna, todos los programas que quepan en una cara de la cinta.

Por otra parte, y dado que cada programa se carga mediante un comando LOAD, aunque utilizado de una forma especial, será válido y se deberá tener en cuenta todo lo dicho para la carga de programas utilizando el comando LOAD.

Vamos a ver a continuación con un ejemplo, el modo de proceder para encadenar tres programas. Hemos escogido tres programas sencillos; sin embargo, todo lo que se indica para ellos es válido para programas de cualquier complejidad.

Los tres programas que vamos a utilizar como ejemplo realizarán las siguientes tareas: el primer programa realizará una presentación por pantalla, indicando de qué se trata el proceso que se realizará a continuación; el segundo programa realizará el dibujo de un triángulo; por último, el tercer programa realizará el cálculo del área del mismo, suponiendo que el usuario le indica los datos de base y altura.

Dado que necesitaremos espacio para grabar los tres programas, utilizaremos una cinta en la que tengamos una cara vacía, de forma que podamos despreocuparnos de la longitud de los programas que vamos a grabar.

El primer paso será conectar adecuadamente el cassette, y colocar la cinta al principio de la cara vacía. Si nuestra grabadora dispone de contador de vueltas, lo pondremos a cero, una vez hayamos rebobinado toda la cinta.

Ahora podremos ya proceder a escribir el primer programa. Este programa se llamará programa «A», y tal como hemos dicho, nos servirá para presentar el proceso que se realizará a continuación. El listado de este programa podría ser:

```
NEW
10 REM CALCULO DEL AREA DE UN TRIANGULO
20 REM ---- 1a. Fase. Presentacion ----
30   CLS
40   PRINT : PRINT : PRINT
50   PRINT "****************************************"
60   PRINT "*                                      *"
70   PRINT "*       AREA DE UN TRIANGULO           *"
80   PRINT "*                                      *"
90   PRINT "****************************************"
100  PRINT : PRINT : PRINT
110  PRINT "       Fase 1. Dibujo del triangulo"
120  PRINT
130  PRINT "       Fase 2. Calculo del area"
140  PRINT : PRINT : PRINT
150  PRINT "Pulse cualquier tecla para continuar"
160  LET A$ = INKEY$ : IF A$="" THEN GOTO 160
```

Compruebe el resultado de este programa escribiendo RUN.

Si lo desea realice las modificaciones que crea convenientes para mejorar la presentación. Por otra parte, recuerde cuál es la utilidad de la instrucción INKEY$ tal como está escrita: el proceso se detiene mientras el usuario no pulse una tecla.

Una vez comprobado el buen funcionamiento del programa, añadiremos la instrucción que nos permitirá cargar automáticamente el siguiente programa. Dado que los tres programas se llamarán A, B y C, en éste, que es el programa A, daremos la indicación de cargar el siguiente, que será el B. La instrucción ha de formar parte del programa, por tanto, se escribirá numerada. Así, escribiremos:

```
170 LOAD "B"
```

Cada programa termina con la instrucción de carga del siguiente

Ahora podemos ya salvar el programa. Recordemos que se debe hacer con la indicación de número de línea. Por tanto, una vez conectada la grabadora y colocada la cinta, dejaremos un par de vueltas en blanco —es aconsejable hacerlo, para facilitar la posterior lectura— y daremos la orden de grabación. Así, escribiremos:

```
SAVE "A" LINE 10
```

«A» es el nombre de este primer programa, y 10 es el número de la línea por la que deberá empezar a ejecutarse.

A continuación deberá verificar que la grabación se haya realizado correctamente, rebobinaremos la cinta hasta el principio y escribiremos:

```
VERIFY "A"
```

o el comando equivalente de verificación en su máquina.

Si la grabación se ha efectuado correctamente, podremos proceder ya a escribir el siguiente programa.

Este programa deberá dibujar un triángulo, de esta forma:

```
*

***

*****

*******

*********
```

Esta forma de dibujarlo es sencilla, pues basta construir líneas de longitud creciente, e irlas imprimiendo. Se puede utilizar cualquier símbolo, aunque en este caso utilizaremos un asterisco (*).

El programa para ello será:

```
NEW
10 REM DETERMINACION DEL AREA DE UN TRIANGULO
20 REM --- 2a. Fase. Dibujo del triangulo ---
 30    CLS : PRINT : PRINT : PRINT : PRINT
 40    LET I = 1 : LET A$="*"
 50    IF I>10 THEN GOTO 100
 60          PRINT A$
 70          LET A$ = A$+"**"
 80          LET I = I+1
 90          GOTO 50
100    PRINT : PRINT : PRINT : PRINT
110    PRINT "Pulse cualquier tecla para continuar";
120    LET A$ = INKEY$ : IF A$="" THEN GOTO 120
```

Compruebe el funcionamiento de este programa escribiendo RUN. La línea 30 borra la pantalla y escribe cuatro líneas en blanco. A continuación —línea 40— se inicia la construcción del bucle que escribe el triángulo: asigna el valor inicial a la variable de control y construye la primera línea. Esta está constituida por un sólo asterisco, que será el vértice del triángulo. El control del bucle se efectúa en la línea 50: cuando se hayan escrito 10 líneas se detendrá el proceso. Ahora entraremos en el bucle, y el primer paso será imprimir la línea. A continuación se construirá la línea siguiente, añadiendo dos asteriscos a la anterior. Incrementaremos por último la variable de control, y saltaremos de nuevo a la línea 50.

De esta forma se irá ejecutando el proceso hasta escribir las diez líneas, con lo cual ya habremos dibujado el triángulo. Ahora se imprimirán cuatro líneas en blanco, y por último el programa esperará que pulsemos una tecla para proseguir.

Una vez comprobado el funcionamiento del programa, añadiremos la instrucción de carga del siguiente programa. Así, escribiremos:

```
130 LOAD "C"
```

Tal como hemos dicho, el programa siguiente se llamará programa C. Ahora podremos ya salvar este programa. En primer lugar haremos avanzar la cinta un par de vueltas —para delimitar claramente la separación de los dos programas, de cara a la lectura posterior de éstos— y a partir de este momento, teniendo las conexiones de forma conveniente, escribiremos:

```
SAVE "B" LINE 10
```

Con lo cual, el programa que tenemos en memoria, se almacenará con el nombre de programa B, y al cargarlo se empezará a ejecutar directamente a partir de la línea 10.

Una vez grabado, deberemos como siempre verificarlo. Rebobinaremos la cinta hasta el principio de este programa, para lo cual nos será útil el contador de vueltas. Si no se dispone de contador desconectaremos momentáneamente el ordenador del cassette, y escucharemos la cinta. De esta forma deberemos oír en primer lugar el sonido —pitido característico— del primer programa, a continuación pasarán las dos vueltas en blanco —no oiremos nada—, y por último volveremos a escuchar el pitido. En este punto empieza el programa B. Situaremos la cinta de forma que el cabezal de lectura esté un poco antes de este punto, conectaremos la grabadora de nuevo al ordenador, y escribiremos:

```
VERIFY "B"
```

o el comando correspondiente en su máquina.

Si la verificación es correcta, podremos ya proceder a escribir el tercer programa. El listado de éste podrá ser:

```
NEW
10 REM DETERMINACION DEL AREA DE UN TRIANGULO
20 REM ----- 3a. Fase. Calculo del area -----
30   CLS : PRINT : PRINT "********** DATOS **********"
40   INPUT "Entre el valor de la Base: ",B
50   INPUT "Entre el valor de la Altura: ",H
60   PRINT : PRINT
70   PRINT "*********** CALCULOS ***********"
80   PRINT
90   PRINT "                BASE X ALTURA "
100  PRINT "       AREA  = -----------------"
110  PRINT "                    2"
120  LET A = B*H/2 : PRINT
130  PRINT "                 ";B;"X";H
140  PRINT "    AREA  = ------------------ = "; A
150  PRINT "                    2" : PRINT : PRINT
160  PRINT "*********** RESULTADO ***********"
170  PRINT
180  PRINT " El valor del area es: ";A
190  PRINT : PRINT : PRINT
200 END
```

Comprobaremos que este programa se ejecute correctamente.

La mayor parte de las instrucciones son de presentación. En primer lugar se borra la pantalla, a continuación se preguntan las dimensiones del triángulo. Y una vez conocidas éstas, se escribe, en el apartado «CALCULOS», la fórmula para determinar el área del triángulo. A continuación se escribirá la misma fórmula, sustituyendo los valores de base y altura, incluyendo el resultado de este cálculo, que se habrá realizado en la línea 120. Por último, se incluye un apartado de «RESULTADO», en que se explicita el valor del área que se ha calculado.

Si el programa funciona correctamente, ya podremos salvarlo, dado que en este caso no hay que incluir ninguna instrucción para encadenar otro programa. Por tanto, haremos avanzar la cinta un par de vueltas, y escribiremos:

```
SAVE "C" LINE 10
```

En el último programa no se debe incluir la instrucción «LOAD»

con lo que almacenaremos el último programa a encadenar.

Deberemos ahora verificar la grabación, por lo que una vez situada la cinta correctamente —tal como hemos hecho con el programa B— escribiremos:

```
VERIFY "C"
```

Si la grabación es correcta, podremos ya comprobar el funcionamiento global de los tres programas. Para ello deberemos en primer lugar rebobinar la cinta hasta el principio. Bastará ahora escribir:

```
LOAD "A"
```

con lo que desencadenaremos la carga y ejecución sucesiva de los tres programas.

Así cuando finalice la carga del primero, automáticamente aparecerá en pantalla lo que muestra la figura 1.

Pulsando ahora una tecla cualquiera, se cargará el siguiente programa, y se empezará a ejecutar directamente. En pantalla veremos ahora un triángulo similar al de la figura 2.

Si ahora pulsamos una tecla cualquiera, se cargará el siguiente programa. En primer lugar se borrará la pantalla, y la máquina nos preguntará los valores de base y altura. Supongamos que damos el valor 4 para la base y 3 para la altura. La pantalla quedará ahora con los datos que muestra la figura 3. Usted puede mejorar la presentación para adaptarla a la pantalla de su ordenador.

Comprobaremos que este programa se ejecute correctamente. En todo

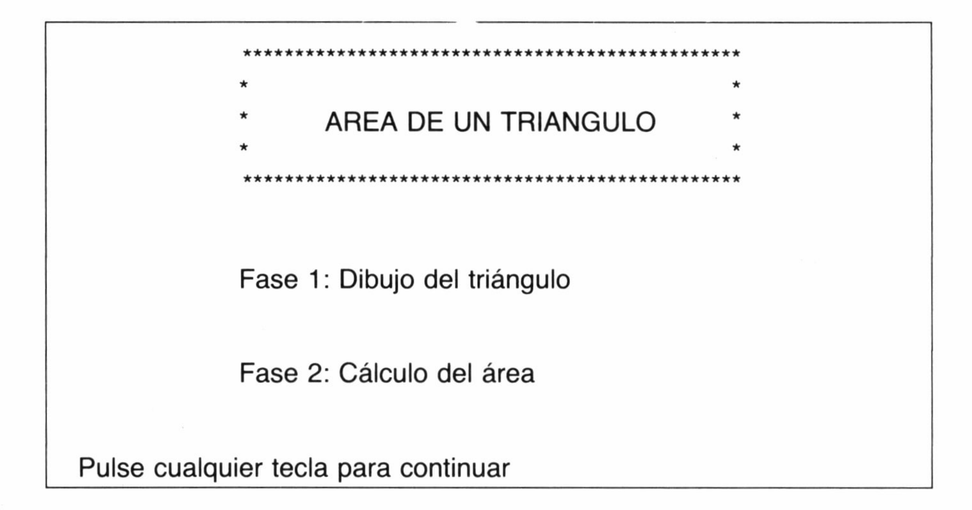


Figura 1 Formato de la pantalla realizada por el programa «A».

caso, puede arreglar la presentación, si en su ordenador no es del todo correcta, adecuándola a los espacios por línea que usted tiene.

Hemos visto pues como se realiza el encadenamiento de programas.

Esto puede hacer un proceso algo más lento, pero siempre resulta tal como ya hemos dicho, más sencillo manipular y comprobar programas cortos. Además siempre resulta más fácil resolver un problema por partes, es decir dividiéndolo en problemas más pequeños.

Los valores de las variables no se conservan en los programas encadenados

Sin embargo, debe tenerse presente que cada vez que se carga un nuevo programa, se pierden los valores de las variables que utiliza el programa anterior. Por tanto, si en un programa complejo se necesita conservar valores a lo largo de toda la ejecución, no se podrá dividir en sucesivos programas encadenados. De todas formas, aun en este caso, es aconsejable escribir el programa por partes, comprobando cada una de ellas y prescindiendo de las demás.

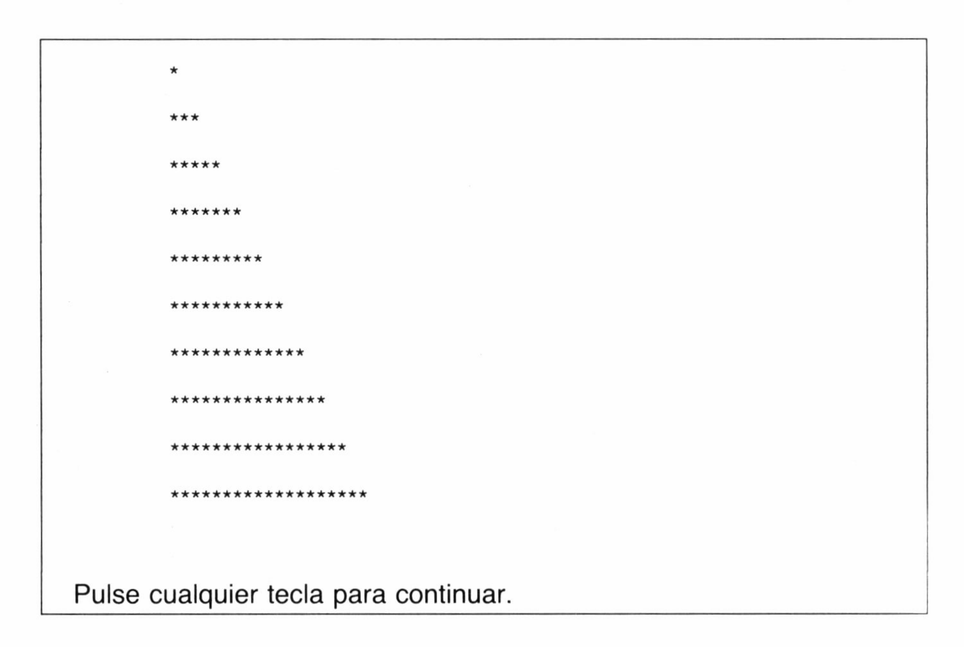


Figura 2 Triángulo trazado por el programa «B».

Figura 3 Presentación de los datos realizados por el programa «C».

```
*********** DATOS ***********

Entre el valor de la Base: 4
Entre el valor de la Altura: 3

*********** CALCULOS ***********
```

$$AREA = \frac{BASE \times ALTURA}{2}$$

–35–

$$AREA = \frac{4 \times 3}{2} = 6$$

```
*********** RESULTADO ***********

El valor del area es: 6
```

En este caso se numerará cada parte de forma adecuada, de manera que después se puedan juntar todas mediante la instrucción MERGE que ya hemos visto. Una buena política es numerar las distintas partes de 100 en 100, o de 1000 en 1000. Así, la primera parte tendrá los números de línea del 10 al 100. La comprobaremos y la grabaremos. La segunda parte la numeraremos del 200 en adelante, y una vez comprobada, se podrá grabar. Las partes siguientes se escribirán con los números 300, 400, etc.

Una vez todas comprobadas, se podrán reunir cargándolas sucesivamente con el comando MERGE, debiéndose comprobar ahora el funcionamiento del conjunto. Cuando se hayan realizado las pruebas necesarias podrá salvarse el programa resultante, se verificará la grabación, y si ha sido correcta, se podrán borrar, si se desea, las distintas partes grabadas previamente por separado.

RESUMEN

El cassette se utiliza para almacenar programas de forma permanente. El proceso se basa en transformar el programa en una serie de impulsos audibles que se pueden grabar en un cassette normal de música.

El cassette puede ser cualquier modelo de grabadora, mejor si es de las más sencillas. En general, los aparatos de alta fidelidad funcionan peor porqué están preparados para ser escuchados por el oído humano y no por el ordenador.

Los ordenadores ya llevan los mecanismos preparados para realizar la conexión.

Otra ventaja adicional es que el cassette permite el intercambio de programas con nuestros amigos que disponen del mismo modelo de ordenador. Las órdenes que dispone el BASIC para manipular la grabación son:

LOAD: Carga un programa desde la cinta a memoria. Si el comando se acompaña de un texto entre comillas, la carga no se efectuará si el programa no fue grabado con aquel nombre específicamente. Si sabemos que el programa no existe puede servir para saber los programas que están grabados en la cinta.

SAVE: Sirve para copiar en la cinta del cassette un programa que reside en memoria del ordenador. Normalmente va acompañado de un texto entre comillas que es el nombre del programa. El número de letras que puede tener el nombre del programa depende de cada ordenador. Una variante de SAVE es incluir después del nombre del programa la palabra LINE seguida de un número. Esta variante permitirá que cuando se cargue el programa, éste se ejecute sin necesidad de teclear el RUN.

VERIFY: Sirve para comprobar que el contenido de la cinta coincide con el contenido de la memoria.

MERGE: Permite cargar un programa mezclándolo con el que se tiene en memoria. El efecto del comando es mantener el programa que tenemos en memoria, añadiendo al mismo las líneas que va leyendo de la cinta del cassette como si lo entraramos desde el teclado. En el caso de que los números de líneas coincidan, el efecto es exactamente el mismo que si se entran por teclado. Cuando se repiten números de línea sólo queda registrada la última línea que se ha entrado.

Finalmente, es posible encadenar programas uno detrás de otro, añadiendo la instrucción de carga en cada uno de ellos al final de proceso y como sentencia numerada. Para que esta ejecución se realice, es necesario que el programa que se carga se haya almacenado con la sentencia SAVE seguida del número de línea.

EJERCICIOS DE AUTOCOMPROBACION

Completar las frases siguientes:

1. El cassette es un periférico del ordenador que permite almacenar los programas de modo ....

2. Las grabadoras de alta fidelidad suelen dar .... resultados para almacenar programas con el ordenador.

3. Los ordenadores ya llevan .... los mecanismos necesarios para hacer las copias de los programas a cassette.

4. El comando .... es el que permite comprobar que el contenido de la memoria y el contenido de un programa en cinta de cassette coinciden.

5. El comando SAVE permite .... los programas en la cinta de cassette.

6. Para añadir unas instrucciones que están grabadas en cinta al programa que estamos realizando se utiliza el comando ....

7. El comando LOAD en muchos ordenadores carga el primer programa que se encuentra en la cinta tal como la tengamos colocada si le sigue un texto ....

8. Cuando se realiza la mezcla de dos programas mediante el comando MERGE la línea que subsiste en el caso de repetición de números de línea es la del programa que está en ....

9. Para cargar y ejecutar un programa sin necesidad de teclear el RUN es necesario utilizar el SAVE seguido del .... y el número de la línea donde se inicia la ejecución.

10. La carga de un programa desde cinta a memoria se hace mediante el comando ....

Encierre en un círculo las alternativas que correspondan a la respuesta correcta:

11. El punto clave para conseguir que una grabación de un programa se realice correctamente en una cinta de cassette es que:

    a) La grabadora sea de alta fidelidad

    b) El volumen esté correctamente ajustado

    c) La cinta sea de calidad superior

    d) El cassette tenga contador de vueltas

12. Para cargar un programa en memoria desde la cinta de cassette es necesario utilizar el comando

    a) LOAD
    b) SAVE
    c) VERIFY
    d) MERGE

13. Para comprobar que el programa en memoria coincide con el programa que contiene la grabadora se utiliza el comando

    a) LOAD
    b) SAVE
    c) VERIFY
    d) MERGE

14. Para mezclar dos programas en la memoria del ordenador, el primero de los cuales ya está en memoria y el segundo está en cinta de cassette, se utiliza el comando

    a) LOAD
    b) SAVE
    c) VERIFY
    d) MERGE

15. Para guardar un programa que tenemos en la memoria del ordenador en el cassette es necesario utilizar el comando

    a) LOAD
    b) SAVE
    c) VERIFY
    d) MERGE

16. Para cargar un programa en la memoria del ordenador y ejecutarlo inmediatamente sin necesidad de teclear el RUN es necesario utilizar el comando

    a) LOAD «rápido»
    b) SAVE «nombre» LINE 20
    c) SAVE «nombre» inmediato
    d) MERGE «A» AND «B» ; AFTER EXECUTE

17. Cuando el programa a cargar desde cinta a memoria no coincide con el nombre que hemos dado, el ordenador realiza las operaciones

    a) Se para la cinta y escribe el mensaje «PROGRAM NOT FOUND»

    b) Continúa con la cinta pero el ordenador da un error que dice «MEMORY FULL»

    c) Rebobina la cinta e intenta la búsqueda con el primer programa que tiene en la cinta

    d) El ordenador se salta el programa y continúa con el siguiente que está en la cinta para ver si coincide con el que buscamos

18. Cuando se realiza un MERGE y hay dos líneas que coinciden en el número la acción que toma el ordenador es

    a) Dar el error «LINE NUMBER MATCHING»

    b) Ignorar la línea que proviene del programa en cassette

    c) Añade un 1 al número de línea del programa que proviene del cassette y los coloca en memoria

    d) Substituye la línea que había en el programa en memoria por la línea del programa en cassette

19. Se debe apretar la tecla REC y PLAY de la grabadora cuando realizamos el comando

    a) LOAD

    b) SAVE

    c) VERIFY

    d) MERGE

20. ¿Cuál de las propiedades siguientes no es característica del cassette?

    a) Posibilidad de intercambio de cintas con un ordenador igual

    b) Los programas no desaparecen cuando desconectamos el ordenador

    c) Los programas se pueden guardar dos veces en la misma cinta

    d) Podemos cargar una parte del programa



8.7 ORGANIZACION DE LA MEMORIA

En las secciones anteriores hemos dicho que si ejecutamos programas largos, o que manipulen gran número de variables, especialmente listas o tablas de gran tamaño, podemos llegar a ocupar toda la memoria del ordenador. Veamos por qué.

Recordemos lo que habíamos dicho en el primer capítulo sobre la memoria de la máquina.

Habíamos visto que se puede considerar la memoria dividida en casillas. Cada casilla tiene el tamaño de un byte (ocho bits). Y se identifica por medio de un número. Este número se denomina «dirección» de la memoria, y tiene un valor comprendido entre 0 y 65535, en los ordenadores personales más corrientes, ya que su valor se almacena en dos bytes, y este número es el máximo que se puede almacenar en 16 bits (2 bytes).

Así, se puede imaginar la memoria como una larga fila de casillas numeradas, cada una con su contenido particular.

Dentro de la memoria se pueden distinguir diversas zonas. Cada zona está dedicada a una tarea concreta, de esta forma se consigue un funcionamiento más rápido y eficaz.

Veamos en primer lugar los tipos de memoria de que dispone el sistema.

Tipos de memoria

Memoria ROM: El contenido de esta memoria es inalterable. Se escribe en el momento en que se fabrica el ordenador, y únicamente se puede leer, pero tal como hemos dicho, no se puede escribir ni modificar. Este tipo de memoria se denomina permanente, pues se mantiene su contenido aunque se desconecte la máquina. ROM es la abreviatura de las palabras inglesas Read Only Memory, que significa «memoria sólo de lectura».

Memoria RAM: El contenido de estas casillas es modificable. Esta memoria es la que se utiliza para trabajar. Este tipo de memoria es no permanente. Sólo se mantiene mientras está la máquina conectada. RAM es la abreviatura de las palabras inglesas Random Access Memory, que significa «Memoria de Acceso al Azar», debido a que se puede acceder —ya sea para leer o para escribir— a una casilla situada en cualquier posición.

Aunque físicamente ambos tipos de memoria son distintos, desde el punto de vista de la organización lógica del sistema se pueden considerar indistinguibles: el conjunto total de memoria es continuo, aunque en realidad esté constituida en parte de ROM y en parte de RAM.

La primera parte la constituye la parte de ROM. En ella se encuentra el «conocimiento» de la máquina. Al fabricar el ordenador se escriben estas memorias; de alguna manera se puede decir que se enseña a la máquina a realizar unas determinadas tareas.

En esta parte se encuentra el intérprete de BASIC. Este no es más que un programa que —tal como su nombre indica— interpreta las órdenes que nosotros escribimos en la máquina en lenguaje BASIC.

La parte de memoria de tipo RAM se puede distribuir de distintas formas. En ella se almacena todo lo que necesita el ordenador para trabajar. Insistimos en que su distribución depende de la máquina.

Vamos a describir de forma muy superficial la finalidad de algunas de las zonas en que se distribuye esta parte de la memoria.

La distribución de la memoria

Así, por ejemplo, una parte contendrá un archivo de presentación de pantalla, que almacena la imagen de televisión, así como los atributos (color, intensidad, etc.) para cada posición de la pantalla.

Se necesita también una parte de memoria reservada para la impresora, no en cuanto al funcionamiento de ésta, sino como almacén de los caracteres que se deban ir imprimiendo. Esto se necesita porque el ordenador no envía a la impresora un carácter cada vez que se tenga que imprimir, sino que los va guardando en un almacén, y los envía en el momento en que éste se llena. De esta forma se consigue un funcionamiento más rápido y eficaz.

Otra parte de la memoria se necesita para almacenar las variables del sistema, que no son las variables definidas por el usuario, sino, la información del estado del sistema en cada momento. Así, esta parte de memoria contiene los límites de cada una de las zonas en que está distribuida la memoria.

Otra zona se dedica a almacenar la información sobre los dispositivos de entrada y salida, es decir, el teclado, la pantalla y la impresora.

Necesitaremos también una parte destinada a almacenar nuestro programa, así como las variables que éste genera.

Existen aún otras divisiones de la memoria, sin embargo, dado que esta distribución es distinta en cada máquina, en el correspondiente capítulo de Prácticas con el microordenador, encontrará cuál es la posición exacta de cada una de estas divisiones, y una explicación más concreta sobre su funcionamiento. Esto le servirá no sólo para conocer internamente la máquina, sino para manipularla si así lo desea, ya que más adelante, en el tomo V, veremos unas instrucciones que nos permitan acceder directamente a una posición concreta de memoria.

La figura 4 representa el mapa de memoria con una posible distribución de la misma.

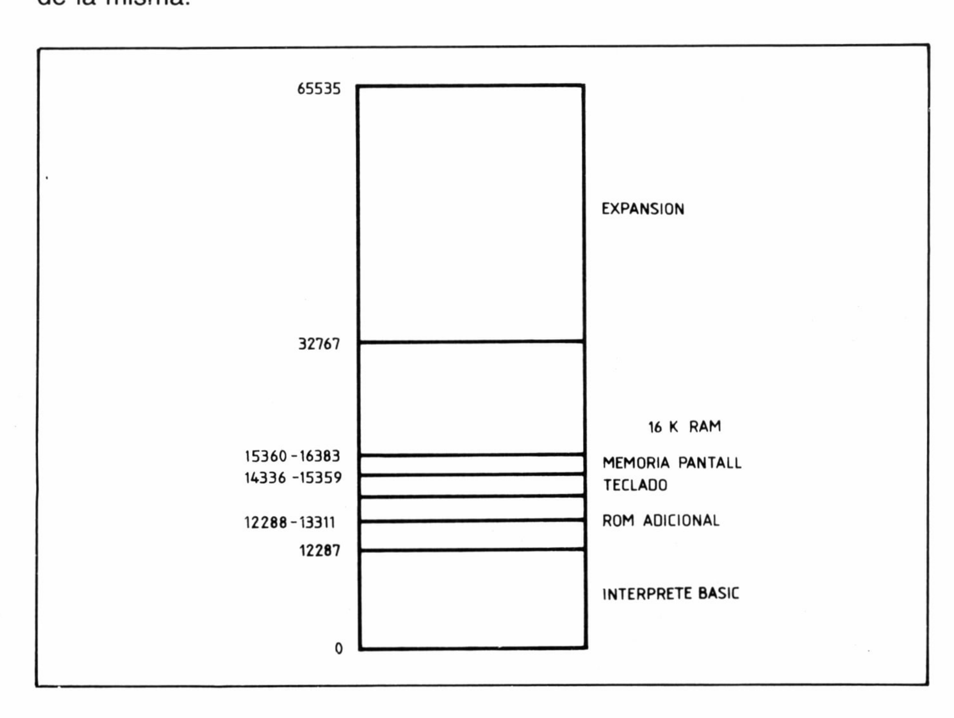


Figura 4 Mapa de memoria.

Recuerde sin embargo, que aunque existe una distribución determinada, dentro de la memoria las casillas se diferencian únicamente por su dirección, sin que exista ninguna otra posibilidad de distinguirlas.

En general las máquinas no disponen de toda la memoria de tipo RAM que son capaces de direccionar, es decir, no poseen —físicamente— las 65535 casillas. Sin embargo estas memorias se pueden incorporar a la máquina sin ningún problema adicional.

Cuando usted adquiere una expansión de memoria lo que hace es incrementar el número de memorias de tipo RAM.

En la siguiente sección veremos una instrucción que nos permite manipular la memoria.

8.7.1 Instrucción CLEAR

Esta instrucción se escribe:

CLEAR

y su ejecución produce el borrado de todo el contenido de la memoria RAM correspondiente a variables.

Su efecto en cuanto a la memoria es el mismo que si escribimos el comando RUN: este comando borra todas las variables antes de iniciar la ejecución del programa.

Anular una variable es distinto a anular su valor

Insistimos en que el efecto de esta instrucción es anular todas las variables. Esto es distinto de anular su valor, es decir, igualarlas a cero. Tenga presente que cuando en un programa aparece una variable, el ordenador la crea, en el sentido de que reserva el espacio para almacenar el valor de esta variable; esto se hace para cada una de las variables, de modo que en un programa se genera una tabla cuyo tamaño dependerá del número de variables que se utilicen. Cada vez que se modifica el contenido de la variable, el nuevo valor sustituye al que ésta tenía en la tabla.

Así, cuando igualamos a cero una variable no la borramos de la tabla, sino que modificamos su contenido haciéndolo igual a cero. Para borrarla efectivamente, utilizaremos la instrucción CLEAR, que anulará toda la tabla que ha sido creada, del mismo modo que lo hace —de forma automática— el comando RUN, antes de iniciar la ejecución del programa.

Los datos que manipula el BASIC son de dos tipos, como ya sabe, variables numéricas y variables textuales, la diferencia más significativa entre estos dos tipos de variables es que las variables numéricas ocupan una longitud en bytes siempre igual, aunque durante la ejecución del programa cambie su valor. En cambio, en el caso de las variables textuales la longitud no está fijada de antemano, y además varía durante la ejecución del programa.

Para manipular correctamente cada uno de estos tipos de variable el BASIC divide la memoria de los datos en dos partes, una para números y otra para textos.

A veces necesitamos programas que utilicen muchos números y pocos textos; otras necesitamos muchos textos y pocos números. La distribución de memoria que tiene el BASIC puede ser inadecuada a nuestro problema.

La instrucción CLEAR permite redistribuir la memoria de números y la de textos

La instrucción CLEAR permite variar esta distribución añadiendo un número detrás de la misma. Por ejemplo:

```
CLEAR 1000
```

indica que debe asignar 1000 caracteres para los textos. El número que sigue a la instrucción CLEAR indica qué espacio de memoria reservamos a los textos en bytes. Cuando usted realiza un CLEAR sin número detrás, la distribución de la memoria es la que tiene establecida el ordenador cuando se conecta a la corriente.

Ciertamente, nuestra capacidad de maniobra no es ilimitada; si escribimos:

```
CLEAR 100000
```

en un ordenador que dispone sólo de 64 Kby, seguro que no se podrá realizar esta redistribución. (Las direcciones no pueden superar el valor 65536.)

La validez del número que está detrás del CLEAR depende en primer lugar de la memoria de que disponemos y en segundo lugar del tipo de ordenador.

En cualquier caso, para conocer el funcionamiento exacto de esta modalidad de la instrucción CLEAR en su máquina, consulte el capítulo de Prácticas con el microordenador.



8.8 LA IMPRESORA

Los programas que hemos visto hasta ahora nos presentan los resultados en la pantalla.

Sin embargo, usted habrá visto que hay ordenadores que imprimen los resultados sobre papel. Hay listas de personal, nóminas, documentos contables, incluso quinielas, que se imprimen con ordenador.

Para escribir estos documentos, hace falta un dispositivo especial que, de una forma genérica, se conoce como impresora.

Aunque es posible que usted no disponga de una impresora, es necesario que conozca las posibilidades que ésta le ofrece, y cómo se utiliza.

La forma de conocerla y manipularla —caracteres o letras especiales, colocación del papel, cambio de la cinta, etc.— dependerá del modelo de impresora del que se disponga.

Para conocer todos estos detalles, deberá consultar el manual de su máquina en particular. Por otra parte, en cuanto a la conexión de la impresora con el ordenador, en el capítulo de Prácticas correspondiente encontrará las indicaciones necesarias para realizarla correctamente.

En esta sección veremos únicamente las instrucciones para manipular la impresora que son más corrientes. Estas instrucciones no son de BASIC estándard, aunque en general se utilizan, tal como las veremos, en la mayoría de ordenadores personales.

Es importante que usted las conozca aunque no disponga de impresora pues, aunque no tenga que utilizarlas, en cualquier caso le ayudarán a comprender mejor las posibilidades de su máquina.

La impresora la podemos utilizar fundamentalmente para dos tareas distintas. Por una parte, podemos imprimir los listados de nuestros programas, de forma que nos resultará más fácil la comprensión y, si es necesario la corrección de los mismos.

Por otra parte, la impresora resultará útil para escribir los resultados de los programas. Estos resultados pueden ser de cualquier tipo: textos, números, dibujos...

En las secciones siguientes se describen las instrucciones a utilizar en cada caso.

8.8.1 La instrucción LLIST

Cuando tenemos un programa en memoria y deseamos ver el listado, conocemos el comando LIST. Este comando, como ya sabemos nos presenta el listado por pantalla.

Para obtener ahora el listado por impresora, escribiremos:

```
LLIST
```

Observe que en este caso hemos escrito dos veces la letra L, a diferencia del comando de listar por pantalla. El efecto de ambos es el mismo, únicamente se diferencian en que en un caso obtenemos el listado por pantalla (LIST), y en el otro lo obtenemos por impresora (LLIST).

Si usted no dispone de impresora, o si no la tiene conectada y ejecuta este comando, observará (dependiendo de su modelo de ordenador) que la máquina queda parada, sin reaccionar ante ninguna tecla. Esto es debido a que se ha enviado la orden de imprimir, y mientras no conectemos la impresora, el ordenador está pendiente de ejecutar esta orden, y no responderá a ninguna otra mientras no haya podido ejecutarla.

En este caso podrá tomar de nuevo el control desconectando la máquina.

Sin embargo, algunas máquinas detectan el problema y lo indican dando un mensaje, sin llegar a producirse el bloqueo.

8.8.2 La instrucción LPRINT

Todo lo que se escribe por pantalla como resultado de nuestros programas, se podrá obtener escrito por impresora si se dispone de ella. Para obtener un resultado en la pantalla, escribimos la instrucción PRINT y a continuación el valor a imprimir.

Si deseamos que este valor se escriba por impresora, podremos hacerlo de la misma manera; para ello utilizaremos la instrucción siguiente, que se escribirá numerada, si ha de formar parte de un programa:

```
LPRINT
```

y a continuación los valores a imprimir, ya sean constantes o variables, de la misma manera que se escriben cuando se utiliza la instrucción PRINT.

Lo único que debe recordar en este caso es que cuando se va escribiendo en la impresora, el papel va avanzando, sin posibilidad de retroceder. Así, mientras que en la pantalla tenemos la posibilidad de movernos hacia adelante o hacia atrás según nos interese (recuerde el posicionamiento del cursor que se realiza mediante el posicionador AT), en la impresora sólo podemos ir hacia delante. Así, dentro de una línea tenemos la posibilidad de movernos únicamente hacia la derecha utilizando el tabulador TAB.

El movimiento de escritura en la impresora siempre es hacia adelante; no es posible retroceder

Para determinar qué es lo que se escribe en la impresora y que es lo que se escribe en la pantalla, el criterio a seguir se basa en la utilidad de lo que se escribe. Así, en principio, las preguntas al usuario y los resultados intermedios bastará que se escriban por pantalla, dado que constituyen información que únicamente interesa al operador. Por otra parte, los resultados finales se escribirán por impresora, dado que éstos son los únicos que a la larga interesan mantener. Es interesante por otra parte que cuando se escriban estos resultados se indique en la misma hoja el significado de los mismos, pues de lo contrario resulta inútil imprimirlos y guardarlos.

Si interesa que un resultado aparezca por pantalla y por impresora se deberán escribir las instrucciones para ambos casos. Veamos un ejemplo.

Supongamos que deseamos escribir un programa que nos imprima la tabla de multiplicar de cualquier número, que indicará el usuario.

El listado para obtener el resultado únicamente por pantalla, podría ser:

```
 10 REM TABLA DE MULTIPLICAR
 20 REM ------------------------
 30   CLS
 40   PRINT "TABLA DE MULTIPLICAR"
 50   PRINT : PRINT : PRINT
 60   INPUT "De que numero desea la tabla: ",A
 70 REM ------------------------------------------
 80   CLS : PRINT TAB(10);"TABLA DEL ";A
 90   PRINT : PRINT
100   LET I = 0
110   IF I>10 THEN GOTO 150
120        PRINT I;" x ";A;" = ";I*A
130        LET I = I+1
140        GOTO 110
150   PRINT : PRINT : PRINT
160   INPUT "Desea continuar (S/N): ",R$
170   IF R$="S" THEN GOTO 30
180   END
```

Si usted dispone de impresora, una vez conectada correctamente, de- berá escribir:

```
LLIST
```

Ahora, si lo desea, puede añadir las instrucciones para obtener el re sultado por impresora. Observe que lo único que interesa que se escrib: es la tabla, no las preguntas que hace la máquina de cara al usuario. Bas tará entonces modificar las líneas 80, 90, 120 y 150 de forma que la orde de escribir por pantalla se modifique para escribir por impresora. Así el pro grama quedará:

```
 10 REM TABLA DE MULTIPLICAR
 20 REM --------------------
 30   CLS
 40   PRINT "TABLA DE MULTIPLICAR"
 50   PRINT : PRINT : PRINT
 60   INPUT "De que numero desea la tabla:",A
 70 REM ------------------------------------
 80   LPRINT TAB(10);"TABLA DEL ";A
 90   LPRINT : LPRINT
100   LET I = 0
110   IF I>10 THEN GOTO 150
120     LPRINT I;" x ";A;" = ";I*A
130     LET I = I+1
140       GOTO 110
150   LPRINT : LPRINT : LPRINT
160   INPUT "Desea continuar (S/N): ",R$
170   IF R$="S" THEN GOTO 30
180   END
```

Si deseáramos además que nos escribiera la tabla por pantalla y po impresora, deberíamos añadir las órdenes correspondientes. Así, al pro grama anterior añadiremos las instrucciones de escribir por pantalla, con l que éste quedará:

```
10 REM TABLA DE MULTIPLICAR
20 REM --------------------
30   CLS
40   PRINT "TABLA DE MULTIPLICAR"
50   PRINT : PRINT : PRINT
60   INPUT "De que numero desea la tabla: ",A
70 REM ------------------------------------
```

```
 75   CLS : PRINT TAB(10;"TABLA DEL ";A
 80   LPRINT TAB(10);"TABLA DEL ";A
 85   PRINT : PRINT
 90   LPRINT : LPRINT
100   LET I = 0
110   IF I>10 THEN GOTO 150
115        PRINT I;" x ";A;" = ";I*A
120        LPRINT I;" x ";A;" = ";I*A
130        LET I = I+1
140        GOTO 110
145   PRINT : PRINT : PRINT
150   LPRINT : LPRINT : LPRINT
160   INPUT "Desea continuar (S/N): ",R$
170   IF R$="S" THEN GOTO 30
180   END
```

De esta forma, al ejecutar el programa obtendremos el resultado por pantalla y por impresora, tal como deseábamos.

Sin embargo, si usted no dispone de impresora o no la conecta, e intenta ejecutar este programa, observará que la máquina queda detenida, de la misma forma que habíamos visto cuando intentábamos el listado por impresora sin tenerla conectada. La razón del bloqueo es pues la misma que en este caso.

RESUMEN

La memoria del ordenador se clasifica en dos tipos según las características de construcción que posea: la memoria ROM y la memoria RAM.

La memoria ROM (del inglés Read Only Memory) es memoria de sólo lectura, cualquier programa será capaz de leer pero no podrá modificar su contenido.

La memoria RAM (del inglés Random Access Memory) es una memoria en la que se puede leer y escribir.

Aparte de esta diferencia (substancial), no hay otra en apariencia, y no se puede distinguir la transición de una memoria a otra.

La memoria ROM se llena en el momento que se fabrica el ordenador, de esta manera se previene que el usuario pueda modificarla por error, y también sobrevivir a la desconexión de la corriente. En la memoria ROM se almacena el programa que sirve para interpretar nuestros comandos en el lenguaje BASIC.

La memoria RAM sirve para almacenar su programa, por eso desaparece cuando se desconecta el ordenador de la corriente. La memoria RAM aparte de almacenar su programa, también almacena otras variables que necesita el BASIC para funcionar y que usted no puede ver. Estas zonas adicionales que utiliza el mismo programa BA-

SIC son la causa de que usted no disponga de toda la memoria RAM para sus cálculos.

Las partes más destacadas de esta memoria que necesita el sistema son la memoria de pantalla y el controlador del teclado, aunque cada sistema puede consumir una parte más o menos grande según las necesidades.

En la figura 4 se muestra un esquema general, que precisamente por lo general no se ajusta a ningún sistema en concreto, pero que todos los sistemas disponen.

Cuando se ejecuta un programa las variables que necesita se van creando en una zona de memoria reservada al efecto. En el momento que consumimos todo el espacio, diremos que hemos agotado la memoria.

La instrucción CLEAR permite que esta zona de memoria para los datos quede libre otra vez, a costa de perder los valores de las variables.

Debe distinguir entre hacer un CLEAR y anular el valor de una variable; en el primer caso, la variable desaparece del sistema, dejando memoria libre; en el segundo caso el valor cero o vacío substituye al previamente establecido, pero el espacio de memoria no se modifica.

En el BASIC hay dos tipos de variables, las textuales y las numéricas. Las variables numéricas ocupan una cantidad de memoria fija, mientras que las textuales evolucionan durante el progreso del programa ocupando cantidades diversas de memoria. Por ello ambos tipos de variables ocupan zonas de memoria distintas. El BASIC permite cierto control sobre la zona reservada a variables numéricas y textuales mediante la instrucción CLEAR seguida de un número que indica el espacio reservado a las variables textuales.

La impresora permite imprimir los resultados de nuestros programas sobre papel, pero para ello es necesario obviamente disponer del dispositivo.

Hay dos tipos de instrucciones:

1) La que permite listar el programa en la impresora, se denomina normalmente LLIST

2) La que permite imprimir resultados de cálculos. Se denomina LPRINT. Su utilización es muy parecida al PRINT, la diferencia más notable es que cuando se utiliza la impresora no es posible retroceder el papel tal como nos permite hacerlo en la pantallaa mediante la instrucción AT.

EJERCICIOS DE AUTOCOMPROBACION

Completar las frases siguientes:

21. La memoria del ordenador que sólo permite su lectura e impide su escritura se denomina ....

22. La memoria RAM permite que se lea y .... sobre ella.

23. Las memorias ROM y RAM sólo se distinguen en el momento de .... sobre ellas.

24. La memoria para dibujar la pantalla es una parte de la memoria RAM que consume el ordenador y que no puede utilizar el ....

25. Cuando se ejecuta un programa BASIC las .... de nuestro programa se van creando en una zona de memoria reservada al efecto.

26. La instrucción .... limpia la zona de memoria destinada a las variables de nuestro programa.

27. En BASIC la zona de memoria de las variables numéricas es distinta a la zona de las variables ....

28. La instrución .... sirve para obtener sobre la impresora el listado de nuestro programa.

29. La instrucción de imprimir un resultado sobre papel es ....

30. Cuando se escribe sobre papel es imposible .... tal como se hace con la pantalla mediante la instrucción de posicionamiento del cursor.

Encierre en un círculo la letra que corresponda a la respuesta correcta.

31. La manera de mantener un programa permanente con el ordenador es mediante

a) La memoria ROM.

b) La memoria RAM.

c) La grabadora de cassette.

d) No se puede.

32. El BASIC de un ordenador personal está normalmente en ROM, pero para sus cálculos necesita

    a) Un trozo de memoria RAM.

    b) Un teclado con la tecla BREAK.

    c) Un cassette especialmente preparado.

    d) No necesita nada.

33. Si la variable N$ de un programa vale «Hola», después de ejecutar una instrucción CLEAR, su valor será:

    a) ""

    b) "Hola"

    c) "H"

    d) No existirá la variable.

34. Para hacer un listado de nuestro programa por impresora debemos utilizar la instrucción

    a) LIST

    b) LLIST

    c) PRINT

    d) LPRINT

35. Cual de las instrucciones siguientes escribirá el mensaje "He terminado" en la impresora y en la columna 12:

    a) PRINT TAB(12);"He terminado"

    b) LPRINT "He terminado"

    c) LPRINT TAB(12);"He terminado"

    d) LLIST TAB(12);"He terminado"



SOLUCIONES DE LOS EJERCICIOS DE AUTOCOMPROBACION

Capítulo 5

1. Decisiones
2. Listar
3. Precisa
4. Complejo
5. Pregunta binaria
6. Bifurcación
7. Finalización
8. Continuar
9. Depurar
10. Detiene
11. F: Detrás del IF debe ir una pregunta binaria.
12. V
13. F: La pregunta binaria mezcla números y textos.
14. F: La expresión es aritmética con variables textuales.
15. V
16. A=5 (8<2 es falso)
17. A=−5 (−2<−8 es falso)
18. A=1, B=2 (B< es falso)
19. A=2.45 (2.45 > 2.5 es falso)
20. A$=«5» y B$=«A» (A$ es distinto de B$)
21. A$=«m» (Las minúsculas son mayores que las mayúsculas)
22. A$=«m» (La M es menor que Z, la instrucción detrás del THEN convierte una letra mayúscula en minúscula, la diferencia de código ASCII entre mayúsculas y minúsculas es 32)
23. A$=«JUAN» (El texto no es vacío)
24. A = 39 (13 es menor que 26)
25. A = −13 (−13 no es menor que −26)
26. Bifurcación
27. IF ... THEN ... ELSE
28. Diseño
29. Simular
30. Salto
31. Inicio

32. común
33. Pruebas y correcciones
34. Natural
35. Corrección
36. *b*)
37. *c*)
38. *d*) La definición en el diseño debe ser completa, en datos, cálculos y resultados.
39. *c*) No es cuestión de longitud sino de funcionalidad.
40. *a*)

Capítulo 6

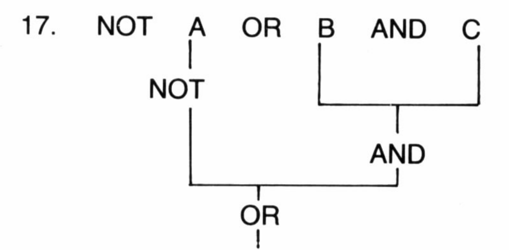
1. lógica
2. unario
3. cambia
4. binarios
5. ciertas
6. falso
7. distintas
8. tabla de verdad
9. lógicos
10. aritméticos
11. c
12. c
13. d
14. a
15. b
16. NOT A AND B OR C

    NOT

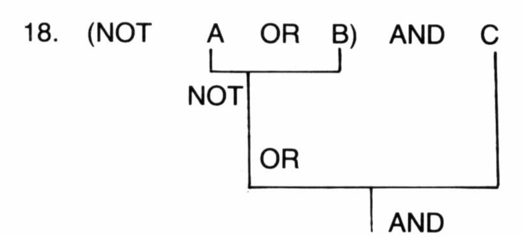
    AND

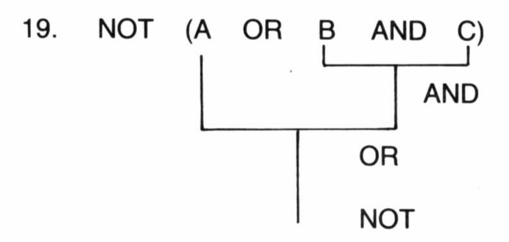
    OR

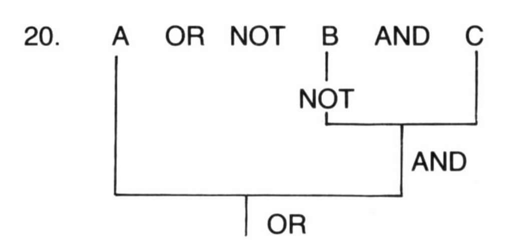
17.

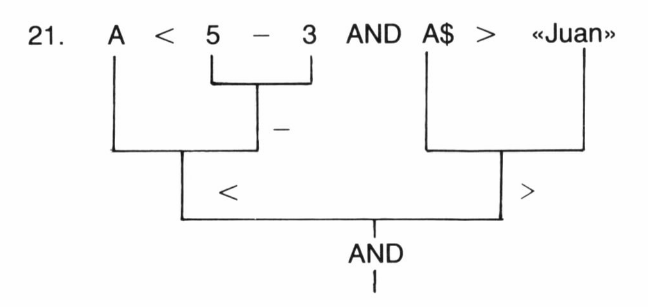


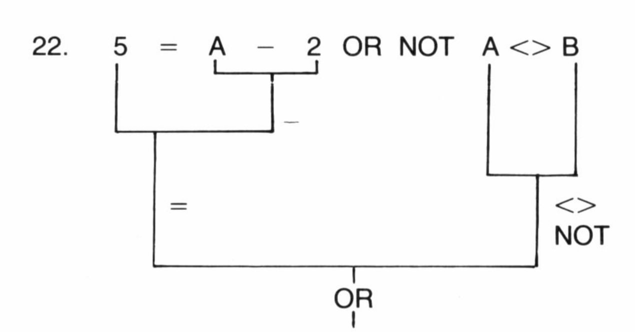
18.

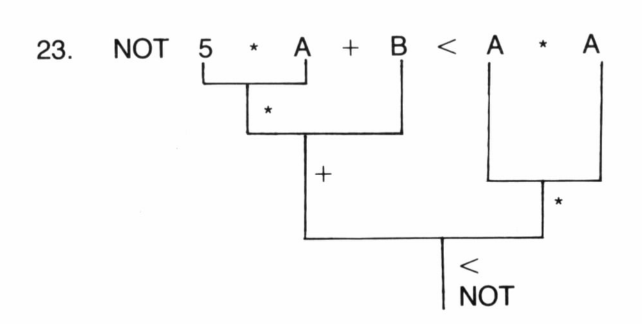



19. NOT (A OR B AND C)
AND
OR
NOT

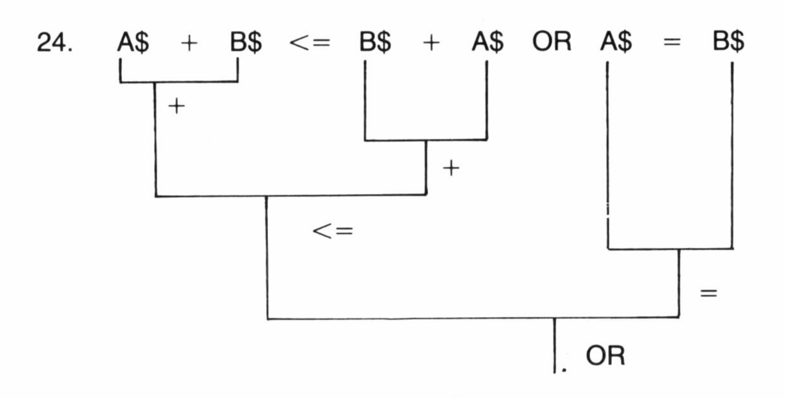

20. A OR NOT B AND C
NOT
AND
OR


21. A < 5 – 3 AND A$ > «Juan»
–
<
>
AND

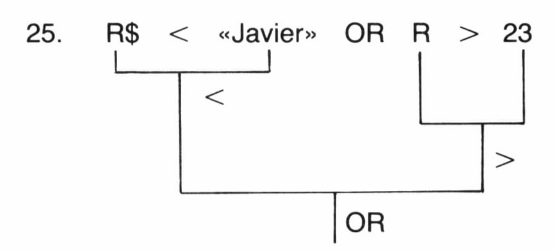

22. 5 = A – 2 OR NOT A <> B
–
=
<>
NOT
OR


23. NOT 5 * A + B < A * A
*
+
*
<
NOT

24.



25.



26. Como hay dos variables, hay cuatro filas

A	B	(A AND B) OR (NOT A AND NOT B)
S	S	S
S	N	N
N	S	N
N	N	S

— Si A AND B es cierto, ya lo es en el primer caso
— Si NOT A AND NOT B es cierto ya lo es, cuando A y B son ambos falsos NOT A AND NOT B es cierto
— Si A es cierto y B falso, o viceversa ni A AND B, ni NOT A AND NOT B pueden ser ciertos.

27. no

28. binarias

29. OR

30. AND

31. contrario

32. distributividad

33. filtro

34. negaciones

35. bits

36. complemento a dos
37. d (Aplicación de las leyes de Morgan)
38. b
39. b
40. a
41. d
42.

A	B	(A AND NOT B) OR (NOT A AND B)
S	S	N
S	N	S
N	S	S
N	N	N

Es idéntica a la del XOR

43.

A	B	NOT (NOT A OR NOT B)
S	S	S
S	N	N
N	S	N
N	N	N

44. 0000 0000 0000 1111 15

1111 1111 1111 0000 Cambio

1111 1111 1111 0001 + 1 Complemento a dos

45. Como el complemento a dos es el cambio de signo, se trata de representar en binario el número 35.

0000 0000 0001 0011

46.
```
7  0111
8  1000
   ____ AND
   0000  0
```

47.
```
2  0010
3  0011
   ____ OR
   0011  3
```

48.
```
5  0101
4  0100
   ____ AND
   0100  4
```

49.
```
10  1010
 5  0101
    ____ OR
    1111  15
```

50.
```
6  0110
4  0100
   ____ AND
   0100  4
```

Capítulo 7

1. Detiene.
2. INKEY$.
3. Vacía.
4. Código ASCII.
5. Menú.
6. Diseño.
7. Básicos.
8. Jerárquica.
9. Documentación.
10. Codificación.
11. b.
12. d.
13. a.
14. c.
15. b.
16. d.
17. d.
18. c.
19. c.
20. b.
21. Detalle.
22. Finalización.
23. Jerárquico.
24. Lista.
25. Símbolos, instrucciones.
26. Dispositivo.
27. Decisión.
28. Regla.
29. Cumple.
30. Alternativas.
31. Tratamiento.
32. Completa.
33. Mismo.
34. Distinto.
35. Condición.
36. b.
37. a.
38. c.
39. d.
40. d.
41. b.
42. d, El criterio de ser rico no es binario.
43. a.
44. c.
45. a.

Capítulo 8

1. Permanente
2. Malos
3. Incorporados
4. VERIFY
5. Guardar
6. MERGE
7. Vacío
8. Cinta
9. LINE
10. LOAD
11. b
12. a
13. c
14. d
15. b
16. b
17. d
18. d
19. b
20. d
21. ROM
22. escriba
23. escribir
24. programador, usuario
25. variables
26. CLEAR
27. textuales
28. LLIST
29. LPRINT
30. retroceder
31. a
32. a
33. d
34. b
35. c

Parte II

PRACTICAS CON EL ORDENADOR

Capítulo 5

ESQUEMA DE CONTENIDO

5.1 INSTRUCCION IF...THEN

Las palabras IF y THEN son palabras clave del BASIC y por tanto en el ZX-SPECTRUM se escriben pulsando una sola tecla. La palabra IF se encuentra sobre la tecla U. La palabra THEN está escrita sobre la tecla G.

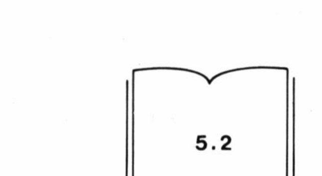
Veamos el programa de prueba

```
10 INPUT B
20 IF B<7 THEN PRINT "CONDICION CUMPLIDA"
30 GOTO 10
```

Pulse la tecla RUN y dele un número, por ejemplo 8. Como no es menor que 7 el ordenador no imprime nada. Ahora dele un 2. Imprimirá CONDICION CUMPLIDA y pedirá un nuevo número.

Hay que detener en el programa el INPUT con un STOP

Para detener el proceso, la tecla BREAK no funciona. Como ya vimos en la lección segunda, la tecla BREAK no opera cuando el programa se encuentra esperando en un INPUT. En este caso se pulsará la tecla STOP (sobre la A) en respuesta a esta instrucción y a continuación ENTER. Ciertamente el programa también se detiene si contestamos un texto en lugar de un número puesto que entonces se produce un error.



5.2 INSTRUCCION END

En el ZX-SPECTRUM no existe la instrucción END. Esta inexistencia no constituye ningún problema si esta instrucción está situada al final del programa puesto que entonces no hace falta ponerla. Como hemos podido comprobar en todos los programas realizados hasta ahora, cuando se acaban las instrucciones, el BASIC da por finalizado el programa y pasa de nuevo el modo inmediato.

El END se sustituye por un salto a una línea inexistente mayor que la última línea

El problema surge cuando la instrucción END debe ir situada en medio del listado. El ZX-SPECTRUM tiene prevista esta circunstancia y, a cambio de no tener esta instrucción, permite emplear un GOTO dirigido a una línea inexistente.

En el BASIC estándar, si un GOTO pasa control a una línea inexistente, se produce un error y el programa se detiene. En cambio en el ZX-SPECTRUM si la instrucción no existe, se pasa control a la línea situada a continuación. Este hecho, que en principio no tendría mayor trascendencia, es importante para sustituir la instrucción END. Colocando en su lugar un GOTO dirigido a una línea que sea *mayor que la última línea* del programa, el BASIC actúa como si hubiera encontrado un END.

Veamos este ejemplo, que cuenta el número de caracteres de un texto:

```
10 INPUT A$ : IF A$="FIN" THEN GOTO 100
20 PRINT "Longitud:";LEN(A$)
30 GOTO 10
```

Si contestamos un texto cualquiera, nos dará el número de caracteres del mismo, incluyendo los espacios. Pero si contestamos la palabra FIN en mayúsculas en respuesta al INPUT de la línea 10, la instrucción IF que se encuentra a continuación pasará control a la instrucción que sigue a THEN. Esta instrucción es un GOTO 100. El BASIC se dirigirá a la línea 100 que no existe, pero ya hemos dicho que en el ZX-SPECTRUM no se produce error por esta causa. Por el contrario intenta encontrar la línea siguiente a la 100. Al no existir tampoco, el BASIC se encuentra en la misma situación que cuando se le acaban las instrucciones de un programa de forma normal. Por tanto da por terminado el programa. En resumen, la instrucción GOTO dirigida a un número de línea mayor que las demás líneas del programa actúa como si fuera una instrucción END.

5.3 INSTRUCCION STOP

La instrucción STOP nos indica siempre en qué línea ha sucedido al ejecutarse

Esta instrucción se encuentra en la tecla A. Como ya sabemos, se usa para detener el programa en un lugar preciso y así poder controlar el buen funcionamiento de dicho programa. Por esta causa, cuando el programa se detiene en un STOP, el BASIC escribe un mensaje informativo indicando en qué instrucción se ha detenido. En el ZX-SPECTRUM, este mensaje tiene la siguiente forma.

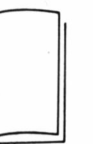
9 STOP statement, 0:0

en donde los dos últimos números (en este caso dos ceros) indican el número de línea y, separado por dos puntos, el número de instrucción dentro de la sentencia. Este mensaje es importante puesto que pueden haber más instrucciones STOP dentro del programa y conviene saber con exactitud en cuál de ellas se ha detenido.

La instrucción STOP puede utilizarse también en lugar de END. El inconveniente radica en que se escribe un mensaje cuando el programa se detiene. Pero si este mensaje no nos causa ninguna molestia, se puede utilizar sin ningún problema la instrucción STOP en lugar de END.

Detrás de la palabra STOP no se admite ningún número o texto a diferencia de algunas variantes del BASIC. De hecho, en el ZX-SPECTRUM, este número o texto es innecesario ya que está destinado a diferenciar cada instrucción STOP de las demás en un programa. En este ordenador, el mensaje que aparece en la parte inferior de la pantalla ya realiza este cometido.

5.4 INSTRUCCION IF CON ELSE



Esta variante de la instrucción IF no existe en el BASIC del ZX-SPECTRUM. En su lugar hay que emplear la instrucción simple IF... THEN en combinación con una o más instrucciones GOTO.

5.5 FASES DE REALIZACION DE UN PROGRAMA

Repase las funciones ASC y MID de la lección 3

En el programa que veremos al estudiar las «fases de realización de un programa» en el capítulo del BASIC estándar, utilizaremos las funciones ASC y MID$ que no existen en el ZX-SPECTRUM. En su lugar, como vimos al estudiar las funciones (el capítulo 3), se emplea la función CODE y el operador de fragmentación respectivamente.

No obstante, por si en este momento no recuerda su utilización le iremos reproduciendo aquellas líneas o partes del programa donde aparecen, para que sepa cómo actuar con su ordenador. Los números de línea se corresponden para facilitar su consulta. En todo caso ésta es una buena ocasión para dar un repaso a lo estudiado en aquel momento.

La primera versión del Paso 2 queda:

```
210 IF LEN(A$))=N THEN GOTO 240
220      LET N1=0
230      GOTO 250
240      LET N1=CODE(A$(N TO N))
250 REM Fin paso 2
```

Las instrucciones cuya ejecución depende de un IF se suelen colocar más hacia adentro para indicar esta dependencia. Esta forma de escribir los programas no mejora el funcionamiento pero sí que ayuda a una mejor comprensión del mismo.

La versión mejorada del Paso 2 queda:

```
210 LET N1=0
220 IF LEN(A$))=N THEN LET N1=CODE(A$(N TO N))
250 REM Fin paso 2
```

A su vez, el Paso 3 se escribirá de la siguiente forma en el ZX-SPECTRUM:

```
260 LET N2=0
270 IF LEN(B$))=N THEN LET N2=CODE(B$(N TO N))
```

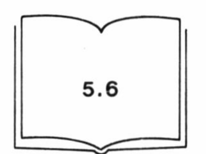
Los pasos comprendidos entre el 4 y el 8, ambos inclusive, no sufren variación alguna. En resumen, en el programa definitivo, las dos únicas líneas que hay que modificar son la 220 y la 270.

```
220 IF LEN(A$))=N THEN LET N1=CODE(A$(N TO N))
270 IF LEN(B$))=N THEN LET N2=CODE(B$(N TO N))
```

Al escribir los resultados (instrucciones 315 y 500), éstos aparecen de forma algo distinta a la que se indica en el texto. Esto ocurre porque la pantalla del ZX-SPECTRUM tiene una anchura de solamente 32 columnas y sólo permite dos zonas de tabulación. Si deseamos tener los resultados tabulados en la misma línea, emplearemos la función TAB. Entonces, las líneas mencionadas anteriormente quedan:

```
305 PRINT N;TAB(10);N1;TAB(20);N2
500 PRINT "FINAL ";N;TAB(15);N1;TAB(25);N2
```

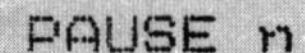
En la fase de cálculo del resultado no hay variación respecto al listado del texto de la lección estándar.



5.6 INSTRUCCION PAUSE

Esta instrucción no forma parte del repertorio del BASIC estándar, si bien es cierto que muchas variantes del BASIC la incorporan. Como su nombre indica (Pausa) significa que la ejecución de un programa se «congela» durante un tiempo para reanudarse seguidamente.

La forma general es:



```
PAUSE n
```

en donde n es un número comprendido entre 0 y 65535. Esta instrucción tiene un contador interno y detiene el programa hasta que dicho contador alcanza el valor de n. La velocidad a la que cuenta este contador depende de la frecuencia de la corriente alterna de la red con que se alimenta el ordenador. En los países europeos, esta frecuencia es de 50 ciclos por segundo. Esto quiere decir que la instrucción PAUSE cuenta a la velocidad de 50 por segundo. Entonces la instrucción

```
PAUSE 100
```

originará una pausa de 2 segundos. El valor máximo, 65535, equivale a una pausa de unos 1300 segundos o 21 minutos.

Para interrumpir el contador, o lo que es lo mismo, para acabar la pausa antes de lo indicado al ordenador, se pulsa una tecla cualquiera. Por tanto hay dos formas de acabar una pausa. La primera es esperar a que el contador llegue a *n* y la segunda es pulsar una tecla durante la espera.

Si el valor de *n* es cero, tiene el significado de espera continuada, es decir, que en este caso sólo se puede interrumpir mediante la pulsación de una tecla.

5.7 PRACTICAS

Con el fin de poder establecer la comparación en programas en que se utiliza la construcción IF... THEN y en aquellos que no se utiliza vamos a realizar las mismas prácticas que en el capítulo 4.

En el ámbito de este capítulo debe apreciar que los programas se han generalizado y adquieren una mayor validez y utilidad práctica pues el control del proceso se hace con más posibilidades ya que se pueden tomar una serie de decisiones que con los conocimientos del capítulo 4 no era posible realizar.

5.7.1 Práctica primera

Paso 1: Definición del problema

Vamos a construir un programa que realice facturas de manera un tanto más general que en el capítulo 4. La generalización más importante consiste precisamente en que allí sólo era posible tener en cuenta tres artículos, mientras que ahora el número de artículos puede quedar indeterminado. Ciertamente esto se acerca más a la realidad de la realización de facturas, pues en general, los consumidores o clientes no compran todos el mismo número de artículos. El modelo será también la figura 1 del primer capítulo del tomo anterior.

Los requisitos que se imponen al programa son:

— El soporte de salida será la pantalla y no el papel. Por tanto, el resultado lo podemos comprobar en la pantalla.

— No es necesario que aparezca la parte de la factura que está con letra de imprenta, sino sólo lo que está a mano.

— Finalmente, el encolumnado tendrá las mismas características que las que se han citado en la práctica primera del capítulo 4, de «Prácticas con su microordenador».

En definitiva, el reparto de columnas para los artículos será:

Columnas	
1-15	Artículos
16	En blanco
17-20	Cantidad
21	En blanco
22-25	Precio unitario
26	En blanco
27-31	Importe

El punto más importante a tener en cuenta aquí es que no es posible establecer una entrada previa de los artículos pues,en realidad no sabemos

de antemano cuántos van a haber en una factura. No nos referimos a que no se conocen los artículos que intervienen en una factura, sino que debemos realizar un programa que admita un número variable de artículos para hacer cualquier tipo de factura.

La estrategia que hay que seguir en este caso consiste en ir escribiendo la factura a medida que se van introduciendo los datos.

Por otra parte, también se generaliza el número de líneas que puede tener la dirección. Es decir, admitimos una dirección con cualquier número de líneas. La realidad siempre impone unos límites razonables, pero en principio el programa no se circunscribe a un número limitado.

Paso 2: Los datos

Los datos que debemos dar al problema son:

a) La fecha (f$)

b) Cada una de las líneas de la dirección (d$)

c) Cada uno de los artículos

c.1) Nombre (n$)

c.2) Cantidad (q)

c.3) Precio unitario (p)

c.4) Importe (i)

d) El tanto por ciento de impuesto que debemos cargar (tpc)

El proceso lo dividiremos ahora en los pasos siguientes:

1. Entrar la fecha y escribirla ya en la factura final.

2. Ir entrando líneas de la dirección hasta obtener una señal para acabar la entrada. Se puede seleccionar como señal la línea vacía. A medida que se entran ya se van escribiendo los resultados en forma de la factura final.

3. Ir entrando los artículos hasta obtener una señal para acabar la entrada. Se puede seleccionar como señal el nombre del artículo vacío. A medida que se entran se va realizando el cálculo.

4. Cálculo del total, entrada del impuesto, cálculo del importe y finalmente el total general.

Debemos señalar que es importante entender el proceso de acabar una entrada mediante una señal de final, pues es un procedimiento muy común en los procesos informáticos.

Detallando más el proceso (y recuerde que esto es un esquema, no un programa):

1. Entrada y escritura de la fecha

```
INPUT "Fecha: " f$
CLS    : Dejar la pantalla limpia
PRINT TAB(9); f$ : PRINT : PRINT
```

2. Bucle de lectura de líneas de dirección y escritura

```
2000  INPUT "Linea de direccion: "d$
      IF d$="" THEN GOTO 2900
      PRINT TAB(15); d$
      GOTO 2000
2900  PRINT : PRINT : PRINT
```

En este bloque se van entrando líneas hasta que se entra una vacía. Esto se consigue pulsando ENTER; el IF nos envía a la instrucción 2900 en donde se imprimen las líneas en blanco. Un esquema alternativo puede ser:

```
2000  INPUT "Linea de direccion: " d$
      PRINT TAB (15); d$
      IF d$<>"" THEN GOTO 2000
      PRINT : PRINT
```

Esta alternativa se basa en que la señal nos escribirá una línea en blanco, con lo que siempre escribimos la línea que entramos; preguntamos a continuación si la línea es vacía. Si no lo es volvemos al INPUT, si lo es sólo escribimos dos líneas en blanco.

3. Entrada de los artículos.

```
3000  LET t=0     Colocamos el total a arrastrar
                  igual a cero.
3010  INPUT "Articulo: " n$
      IF n$="" THEN GOTO 4000
      INPUT "Cantidad: " q : INPUT "Precio unitario: "p
      LET i=p*q : LET t=t+i
      GOTO 3010
```

Tenga en cuenta que en este esquema sólo ponemos de manifiesto la estructura de control y no todos los detalles. Estos los acabaremos de realizar en el momento de escribir el programa. Cuando queramos dejar de escribir artículos, se entra una línea vacía, pulsando ENTER.

4. La idea del cálculo total es prácticamente la misma que en la versión anterior.

Paso 3: Escritura del programa

Ahora teclee el texto del programa de la figura 1 con mucha atención y procurando reconocer en el mismo cada uno de los comentarios que vamos haciendo. Insistimos en que se tome el tiempo y la tranquilidad necesaria para que la práctica rinda los beneficios que esperamos.

En este caso hemos separado los bloques mediante una numeración generosa, cada bloque empieza en las líneas 1000, 2000, 3000 y 4000 respectivamente.

El listado se presenta en la figura 1.

```
1000 REM ------------- Entrada de la fecha
1010   INPUT "Fecha: ";f$
1020   CLS : PRINT TAB (9); f$
2000 REM ----------- Entrada de las lineas de direccion
2010   INPUT "Linea de direccion: ";d$
2020   PRINT TAB(15);d$
2030   IF d$<>"" THEN GOTO 2010
2040   PRINT : PRINT
3000 REM ----------- Entrada de los articulos
3010   LET t=0
3020   INPUT "Articulo: ",n$
3030   IF n$="" THEN GOTO 4000
3040   INPUT "Cantidad: ";q
3050   INPUT "Precio unitario: ";p
3060   LET i=p*q : LET t=t+i : REM Calculo importe
                                acumulacion del total
3070   LET l=LEN n$
3080   LET b$="..............."
3090   LET b$(1 TO l)=n$
3100   PRINT b$;" ";
3110   LET a$=STR$(q)
3120   LET l = LEN(a$)-1
3130   LET b$="    "
3140   LET b$(4-l to 4) = a$
3150   PRINT b$;" ";
3160   LET a$ = STR$(p)
3170   LET l = LEN(a$)-1
3180   LET b$ = "    "
3190   LET b$(4-l TO 4) = a$
3200   PRINT b$;" ";
3210   LET a$ = STR$(i)
3220   LET l = LEN (a$) - 1
3230   LET b$ = "     "
3240   LET b$(5-l to 5) = a$
3250   PRINT b$;" "
3260   GOTO 3020
4000 REM ---------------- Escritura de totales
4010   PRINT TAB(26);"----"
4020   PRINT TAB(19);"Total..";
4030   LET a$ = STR$(t)
4040   LET l = LEN(a$)-1
4050   LET b$= "     "
4060   LET b$(5-l TO 5)=a$
4070   PRINT b$;" "
4075   INPUT "Impuesto: ";tpc
4080   PRINT : LET i=INT (t*tpc/100+0.5)
4090   PRINT TAB(15);"Impuesto ";STR$( tpc);" ";
4100   LET a$=STR$(i)
4110   LET l=LEN(a$)-1
4120   LET b$="     "
```

Figura 1 Programa para hacer facturas

```
4130    LET b$(5-l TO 5)=a$
4140    PRINT b$;" "
4150    PRINT TAB(26);"-----"
4160    LET t=t+i
4170    LET a$=STR$(t)
4180    LET l=LEN(a$)-1
4190    LET b$="     "
4200    LET b$(5-l TO 5)=a$
4210    PRINT TAB(26);b$;" "
4220    PRINT TAB(26);"====="
```

En las líneas 3110 hasta la 3250 realizamos la escritura de las tres columnas, la de la cantidad, el precio unitario, y el total. De hecho cada escritura consume un bloque muy parecido a la columna siguiente, observe los bloques 3110-3150, 3160-3200 y 3210-3250.

Es posible escribir esta parte del programa mediante un único bloque que se repite. En efecto, la escritura que se repite sería: (no escriba todavía)

```
        LET col=1
3110        Escribir el numero asociado a la columna
            LET col=col+1
            IF col<=3 then goto 3110
```

Hemos añadido la variable col que es la que controla el número de columna que estamos escribiendo.

Sin embargo, en cada una de las columnas, escribimos números distintos e incluso tienen longitudes distintas, en el precio y la cantidad tenemos 4 columnas, y en cambio tenemos 5 columnas para el importe. Para tener en cuenta estas diferencias debemos modificar el código de la manera siguiente:

```
        LET col=1
3110    IF col=1 THEN LET lt=4 : LET num=q
        IF col=2 THEN LET lt=4 : LET num=p
        IF col=3 THEN LET lt=5 : LET num=i
            Escribir num con la longitud de columnas lt
            LET col=col+1
            IF col<=3 THEN GOTO 3110
```

Observe que de los tres IF, sólo se pasa por una parte THEN, pues sólo hay una respuesta afirmativa, ya que col vale o 1, o 2, o 3, pero no vale ningún otro valor, y cada IF pregunta por un único valor.

En resumen: Se han introducido tres variables

col : Indica en qué columna escribo

num : Contiene el número que quiero escribir

lt : Contiene la longitud de la columna en la que hay que escribir num.

que permite controlar la escritura encolumnada de forma repetitiva.

El último problema a resolver consiste en escribir el número propiamente. Lo primero que haremos es calcular el número de blancos a añadir y lo colocaremos en la variable nb. El código es

```
LET a$ = STR$(num)
LET nb = lt-LEN(a$)
```

En estos momentos nb contiene el número de blancos que hay que añadir por la izquierda para que a$ tenga justamente la longitud lt.

El último paso consiste en añadírselos con una instrucción como la siguiente:

```
3200 IF nb>0 THEN LET a$=" "+a$ :
     LET nb=nb-1 : GOTO 3200
```

en ella se añade un blanco a a$ con la operación de unión de textos (a$=» «+a$) se descuenta en uno el número de blancos a añadir (nb=nb−1) y se envía otra vez a la pregunta (goto 3200).

Observe que si a$ es mayor que lt, un número con demasiadas cifras, la adición de blancos no se realiza pues nb es un número negativo y nunca es afirmativa la respuesta nb>0. En la versión anterior esto nos daba un error. En esta versión esto no da error, pero nos escribirá mal encolumnado.

Por lo tanto, podremos substituir las instrucciones 3110 a 3250 por el siguiente código:

```
3110 LET col=1
3120 IF col=1 THEN LET lt=4 : LET num=q
3130 IF col=2 THEN LET lt=4 : LET num=p
3140 IF col=3 THEN LET lt=5 : let num=i
3150 LET a$=STR$(num) : LET nb=lt-LEN (a$)
3160 IF nb>0 THEN LET a$=" "+a$ : LET nb=nb-1:GOTO 3160
3170 PRINT a$;" ";
3180 LET col=col+1
3190 IF col<=3 THEN GOTO 3120
3200 PRINT
```

El PRINT de la instrucción 3200 se ha añadido porque en el interior del proceso repetitivo escribimos acabando con un punto y coma, es decir, sin cambiar de línea. En el momento que ya hemos escrito los tres números, debemos colocar un PRINT sin punto y coma para saltar a la línea siguiente.

Para ejecutar el programa en esta nueva versión, no se olvide borrar las líneas 3210 a 3250 inclusive.

4. Pruebas:

El programa ya se ha dividido en bloques para que se pueda probar con comodidad. A medida que entran los bloques, señalados por el cambio

de millar de las líneas, se pueden ir probando para ver si su funcionamiento es correcto.

Los datos que puede entrar son exactamente los mismos que en el capítulo de «Prácticas con su microordenador».

Sólo hay que observar que respecto al cuarto juego de datos, cuando se entra el precio de 125.7 no dará error en la segunda versión de la práctica propuesta, pero se perderá el encolumnado. En este caso, el número de blancos a añadir es menor que cero y en la línea 3160 no se ejecutará la parte del THEN, ya que la respuesta será NO a la pregunta inicial.

5.7.2 Práctica segunda

Paso 1: Definición del problema

El objetivo de este programa es una generalización para la elaboración del coste de un plato de un restaurante, visto en la práctica tercera del capítulo 4.

En definitiva el coste de cualquier cosa es la suma de los componentes, y el coste de cada componente se obtiene multiplicando la cantidad que interviene por el precio unitario. Como puede deducir, el cálculo de una factura y el cálculo de un coste se parecen bastante. No obstante en esta práctica seremos mucho más escuetos en su realización.

La idea del programa consiste en encadenar problemas hasta que hagamos la acción positiva de terminar el programa.

Debemos empezar limpiando la pantalla y colocando una cabecera como la siguiente:

Producto Cantidad Precio Valor

A continuación se deben entrar cada uno de los componentes. Esto significa un nombre, una cantidad y un precio.

Debemos calcular cuánto es el valor y escribirlo en la columna de valor, acumulándolo en forma de total. Para evitar los decimales redondearemos el valor a las unidades.

Para finalizar la entrada de componentes, utilizaremos la señal de que el nombre del componente sea vacío; es decir pulsando sólo ENTER.

Una vez finalizada la entrada se escribirá el total. El resultado final debe tener la forma como indica la figura 2.

Col 1 Producto	**Col 11 Cantidad**	**Col 20 Precio**	**Col 27 Valor**
Calabacín	0.200	70	14
Espinacas	0.080	100	8
Mantequilla	0.025	370	9
		Total	31

Figura 2 Formato de salida de los resultados

Paso 2: Datos

Es necesario llevar un total acumulado, que denominaremos t. Para cada componente se precisa el nombre (a$), el precio (p) y la cantidad (q). También se precisa el valor (v) que obtendremos por redondeo del producto precio por cantidad.

El esquema del programa es el siguiente:

Paso 1 Pantalla inicial limpia con cabecera

Paso 2 Entrada y cálculo de componentes

Hasta que el nombre del componente sea nulo (vacío)

Paso 3 Escritura del total

Paso 4 ¿Quiere continuar con otro cálculo?

SI ———> Retornar al paso 1

(Cualquier otra cosa) ———> Acabar el cálculo

Paso 3: Codificación

Con todos los datos que le hemos dado debería intentar realizar un programa antes de estudiar la versión que nosotros le ofrecemos. Insistimos otra vez en que su programa puede ser distinto y, sin embargo, satisfacer todos los requisitos que le hemos dado.

No obstante recuerde que debe estudiar también nuestra versión del programa y compararla con la suya. Nuestra versión se la ofrecemos en la figura 3.

```
1000 REM Paso 1
1010   LET t=0 : CLS
1020   PRINT "Producto  Cantidad  Precio Valor"
1030   PRINT "--------------------------------"
2000 REM Paso 2
2010 INPUT "Nombre del producto: ",a$
2020   IF a$="" THEN GOTO 3000
2030   INPUT "Cantidad y precio: ";c,p
2040   LET v=INT(c*p+0.5) : LET t=t+v
2050   LET l=LEN(a$)
2060   IF l<10 THEN LET a$=a$+".": LET l=l+1 : GOTO 2060
2070   PRINT a$(1 TO 10);" ";c;TAB(20);p;TAB(27);v
2080   GOTO 2000
3000 REM Paso 3
3010   PRINT TAB(27);"-----"
3020   PRINT TAB(19);"Total...";t
3030   PRINT TAB(27);"====="
4000   INPUT "Otro calculo ";a$
4010   IF a$="s" THEN GOTO 1000
4020   IF a$="S" THEN GOTO 1000
```

Figura 3 Programa para calcular los costes de un producto

Unicamente un comentario a las instrucciones 2050 y 2060, debido a que la columna de producto contiene únicamente las posiciones; si el nombre es más corto, lo alargamos a 10 posiciones añadiéndole puntos; por el contrario, si es más largo, lo cortaremos. La instrucción 2050 calcula la longitud de a$ y la instrucción 2060 alarga el nombre de a$ con puntos (a$=a$+«.») actualizando la nueva longitud (l=l+1) y repitiendo el proceso (GOTO 2060) hasta que la longitud sea 10. En la instrucción 2070 sólo se escriben los 10 primeros caracteres de a$.

5.7.3 Práctica tercera

Paso 1: Definición del problema

Se trata de un programa que intenta adivinar un número que usted ha pensado y que está comprendido entre 1 y 100. El proceso para adivinarlo consiste en dar un número e ir preguntando si es mayor, igual o menor. Si usted responde con un igual es que la máquina ha acertado el número. Si responde con mayor o menor la máquina le presentará otro número. En el momento que lo acierte le informará de cuántas veces ha realizado la operación.

Paso 2: Datos

Los datos fundamentales en este problema es el rango en que puede estar el número que usted ha pensado. Claro está que inicialmente este rango va de 1 a 100. El margen más pequeño se llama parte izquierda y se utiliza la variable *iz* para designarlo, y al más grande se le llama derecha y para él utilizaremos la variable *de*. La estrategia del juego consiste en dar el número de la mitad del rango que tenemos, si no se ha acertado el número la máquina sabrá que está en la mitad superior o en la inferior, con lo que actualizaremos el valor del rango y volveremos a empezar. Por ejemplo, si consideramos el rango inicial, es decir, de 1 a 100, el mejor número a preguntar es el 50, la respuesta admite las alternativas siguientes:

a) Sea igual: Ya se ha acertado el número.

b) Sea mayor: Esto indica que el número que la máquina quiere adivinar está entre 51 y 100.

c) Sea menor: Esto indica que el número que la máquina quiere adivinar está entre el 1 y el 49.

Una vez que hemos obtenido el nuevo rango, lo volvemos a subdividir y realizamos de nuevo la pregunta.

El proceso finaliza cuando se ha adivinado el número.

Si el que contesta le intenta hacer trampa, la máquina lo podrá averiguar, porque detectará un rango que tiene el extremo inferior mayor que el extremo superior: nunca es posible que se dé este caso, si las contestaciones son correctas. Aparte de estos datos fundamentales, precisaremos

de un contador de *n* para saber cuántas veces repetimos el proceso y unas variables para introducir el resultado de la pregunta (a$).

En líneas generales, el programa debe constar de los bloques siguientes:

Paso 0: Inicialización del rango

Paso 1: Cambiar el rango y repetir el paso 1 o salirse según

a) Se ha acertado: ir al Paso 2

b) Se ha hecho trampa: ir al Paso 3

Paso 2: Informar qué número de veces se ha tardado en adivinar el número e ir al Paso 4

Paso 3: Informar del intento de hacer trampa

Paso 4: Preguntar si se quiere jugar otra vez.

El paso más difícil es el Paso 1.

Se inicia este paso preguntando si el rango es correcto o bien se ha hecho trampa; la pregunta es

IF iz>de THEN ir al paso 3

(Recuerde, *iz* contiene el extremo inferior del rango y *de,* el extremo superior)

Una vez se ha determinado que el rango es correcto, se calcula el valor más cercano al medio, decimos más cercano porque si el rango contiene un número par de elementos no existe un número medio. Tomaremos por convenio el número que está más cerca del centro y del extremo inferior. Esto se realiza mediante la instrucción

ac = INT((iz+de)/2)

En esta instrucción lo primero que se calcula es el promedio (iz+de)/2 que puede dar un número fraccionario, por ejemplo, en el rango inicial (1+100)/2 da 50.5; debido a que tenemos un número par de elementos del 1 al 100, ambos inclusive. La utilización de la función INT elimina los decimales suprimiéndolos, por lo tanto, tomaremos el valor más pequeño de los que están en el centro.

A continuación preguntaremos si este número que hemos calculado es el correcto, la respuesta tiene tres alternativas:

a) Que sea efectivamente el buscado, saldremos de este paso

b) Que sea menor, entonces cogeremos el extremo superior del rango y lo rebajaremos al número que se ha presentado menos 1

de = ac−1

c) Que sea mayor, entonces cogeremos el extremo inferior del rango y los aumentaremos al número que se ha presentado más 1

$$iz = ac+1$$

En las dos últimas alternativas, aumentaremos en uno el número de veces que hemos preguntado y reanudaremos el proceso con el nuevo rango obtenido.

Paso 3: Codificación

El listado se le presenta en la figura 4. De todas maneras debería intentar hacerlo con las explicaciones que le hemos dado y comparar su trabajo con la versión que le ofrecemos.

```
  10 LET iz=1 : LET de=100 : LET n=1
1000  IF (iz>de) THEN GOTO 3000
1010  LET ac=INT((iz+de)/2)
1015  CLS
1020  PRINT "El numero puede ser: ";ac
1030  PRINT "El numero que ha pensado es: "
1040  PRINT "       Mayor (>)"
1050  PRINT "       Igual (=)"
1060  PRINT "       Menor (<)"
1070  INPUT a$
1080  IF (a$="=") THEN GOTO 2000
1090  IF (a$=">") THEN LET iz=ac+1 : GOTO 1120
1100  IF (a$="<") THEN LET de=ac-1 : GOTO 1120
1110  GOTO 1070
1120  LET n=n+1 : GOTO 1000
2000  PRINT "Lo he acertado en ";n;" veces"
2010  GOTO 4000
3000  PRINT "Lo siento me ha enganado"
4000  INPUT "Quiere jugar otra vez?";a$
4010  IF a$="s" THEN GOTO 10
4020  IF a$="S" THEN GOTO 10
```

Figura 4 Programa para acertar un número comprendido en un intervalo determinado.

Hemos tenido la precaución de numerar las líneas de tal manera que la cifra de los millares indica a qué paso pertenece

Paso 5: Pruebas

Le sugerimos que pruebe con los números 23, 77, 1, 99, 100

Intente también hacer trampas, basta que introduzca una respuesta incorrecta, por ejemplo, cuando es mayor contestar que sea menor.

Finalmente, cambie la instrucción 10, el límite del extremo superior póngalo igual a 1000, por ejemplo, y compruebe que también funciona; ahora naturalmente, el problema va a tardar más.

Es interesante que estudie bien este ejemplo, pues es la base de muchos algoritmos útiles en programación.

Capítulo 6

ESQUEMA DE CONTENIDO

- Objetivos
- Los operadores lógicos
- Los operadores lógicos sobre números
- Práctica 1. Una sumadora binaria
  - Definición del problema
  - Diseño
  - Sumador propiamente dicho
- Práctica 2. La construcción del filtros
  - Definición del problema
  - Entrada de datos
    - Diseño
    - Escritura del programa
    - Pruebas
  - Selección de información
    - Diseño
    - Escritura del programa
    - Pruebas
  - Aplicación

6.1 LOS OPERADORES LOGICOS

En el ZX-Spectrum los operadores lógicos disponibles son el NOT, el AND y el OR, están situados respectivamente en las teclas *S, Y, U* y no existe el operador XOR.

El valor del SI en el ZX-Spectrum es el 1.

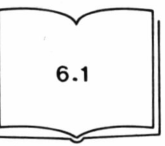
El ZX-Spectrum no presenta variación respecto al comportamiento de los operadores lógicos si actúan sobre variables lógicas; es decir, el funcionamiento de la lección estándar es correcto hasta el apartado 6.6. De todas maneras debe estudiar este apartado con atención; cuando lo finalice estará en condiciones de comprender mejor cómo actúa el ZX-Spectrum.

La única diferencia está en que el operador XOR no lo puede utilizar, pero una vez haya estudiado el apartado 6.3 podrá utilizarlo sustituyendo el operador XOR por la expresión

(NOT A AND B) OR (A AND NOT B)

De momento, no intente entender esta expresión, cuando haya estudiado el apartado 6.3 entenderá su significado.

Por otra parte, cuando en el capítulo estándar escriba los programas de la tabla de verdad de la propiedad de distribuidad y la tabla de verdad de las leyes de Morgan, le advertimos que en las líneas 20, 30, 40 y 50 no utilice las teclas de los operadores AND y OR, sino escríbalos letra por letra. Si utiliza estas teclas en la cadena de caracteres se codifica un solo número y a la hora de salir el resultado por pantalla serán 5 para el AND (las tres letras y los dos blancos) y 4 para el OR. De todas formas puede probarlo de las dos maneras y observar la diferencia.



6.2 LOS OPERADORES LOGICOS SOBRE NUMEROS

Como ya sabe la representación del SI en el ZX-Spectrum es con un 1; esto hace que el comportamiento de los operadores lógicos sobre los números difiera notablemente del explicado en la lección del Basic estándar.

El criterio general que debe seguir es que en el ZX-Spectrum la condición que expresa el NO o el SI se parece más a cero y distinto de cero respectivamente, que propiamente a operaciones lógicas sobre la representación binaria de los números.

a/ Caso del NOT.

El NOT en el ZX-Spectrum da cero si se aplica a un número distinto de cero y uno si se aplica al cero.

Como ejemplo haga los siguientes PRINT inmediatos:

```
PRINT NOT 1          dara 0
PRINT NOT 25         dara 0
PRINT NOT 234        dara 0
```

```
PRINT NOT -3                     dara 0
PRINT NOT 0                      dara 1
```

Vea, por lo tanto, que no hace ninguna operación con los bits, simplemente da de resultado 1 si el número es cero, o da de resultado cero si el número es distinto de cero.

No puede ser de otra manera pues la representación del SI con un 1 impide hacer las operaciones con los bits.

b/ Caso del AND

En el caso del AND el resultado es el primer número si ambos números son distintos de cero y cero en los demás casos.

Observe que basta que uno de los números sea cero para que el resultado sea cero. Veamos algunos ejemplos:

```
PRINT 0 AND 327                  dara 0.
PRINT 0 AND -12                  dara 0.
PRINT 13 AND 0                   dara 0.
PRINT 0 AND 0                    dara 0.
```

Cuando los dos números son distintos de cero, que es el caso más complicado, el resultado es el primero de los dos, es decir, el resultado no es simétrico. Veamos unos cuantos PRINT inmediatos

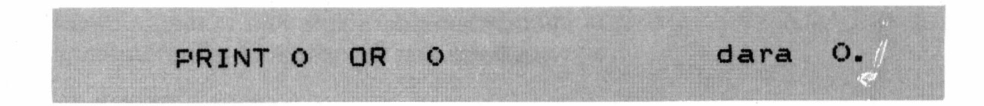
```
PRINT 25 AND 30                  dara 25.
PRINT 30 AND 25                  dara 30.
PRINT -1 AND 2                   dara -1
PRINT -17 AND -13                dara -17.
PRINT -13 AND -17                dara -13.
```

observe que no da lo mismo PRINT 25 AND 30 que 30 AND 25, ni tampoco −17 AND −13 y −13 AND −17.

c/ Caso del OR.

El comportamiento del OR es un poco más complicado.

En un primer caso diremos que sólo da cero cuando los dos operandos son cero, por ejemplo.



```
PRINT 0  OR  0                   dara  0.
```

En un segundo caso, si el primer número es cero y el segundo no lo es, el resultado es 1. Por ejemplo,

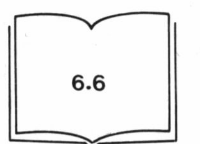
```
PRINT  0 OR  2                    dara  1.
PRINT  0 OR 34                    dara  1.
PRINT  0 OR -5                    dara  1.
PRINT  0 OR 10                    dara  1.
```

En un tercer caso, si el primer número es distinto de cero el resultado es este número, sin importar lo que valga el segundo. Por ejemplo

```
PRINT  2 OR  0                    dara  2.
PRINT 34 OR  2                    dara  34.
PRINT -5 OR  0                    dara  -5.
PRINT 10 OR 23                    dara  10.
```

Las reglas para el NOT, el AND y el OR entre números obedecen a unas reglas que no se ajustan a ninguna relación teórica lo suficientemente importante para que tengan algún valor en programación. Mientras que la teoría que hemos expuesto en la lección del BASIC estándar tiene aplicación en la programación avanzada de la teoría de conjuntos.

Por esta razón, le rogamos encarecidamente no utilice estas reglas últimas en sus programas. No serán transportables a otros ordenadores. Controle, por lo tanto, que todas las variables que aparezcan en expresiones lógicas sean variables booleanas o lógicas. En resumen, este apartado 6.2 que acaba de estudiar, tiene como única finalidad el que usted conozca cómo se comporta su ordenador. Sin embargo, insistimos no tenga en cuenta este comportamiento en sus programas, pues los hará totalmente incompatibles con otras versiones del BASIC. Guíese más bien por lo que estudiará a continuación en la lección estándar.



6.3 PRACTICA 1. UNA SUMADORA BINARIA

La construcción física de los ordenadores se basa en el ensamblaje de distintos circuitos lógicos. Circuitos lógicos significan, aquellos circuitos que realizan las operaciones lógicas sobre entradas eléctricas que sólo pueden tener dos valores, conducir o no conducir, y producen una salida que es a su vez una señal eléctrica lógica.

En realidad, todas las operaciones aritméticas que realiza un ordenador se hacen mediante el cálculo lógico.

La pretensión de esta práctica es mostrar, por una parte cómo funciona el circuito aritmético más sencillo, el sumador, y, por otra parte practicar con las propiedades de los operadores lógicos.

Vamos a imaginar un ordenador que tiene un sumador muy sencillo, concretamente sólo suma números binarios de dos cifras y puede dar el resultado con tres cifras binarias. Veamos el ejemplo siguiente:

```
   1 0
 + 1 1
 -----
 1 0 1
```

Los sumadores de los ordenadores reales no dan más cifras que las de los sumandos, pero siempre, este último dígito queda almacenado en algún elemento de la UCP y el programa lo puede averiguar.

6.3.1 Definición del problema

El procedimiento de sumar un dígito en binario es bien sencillo, la tabla de sumar que se debe recordar es la siguiente:

0 + 0 = 0
1 + 0 = 1
0 + 1 = 1
1 + 1 = 10 (el resultado es 0 y llevo una)

Sólo debemos preocuparnos en sumar dos números. Cuando se suman más, se toman los dos primeros y se suman; el resultado se suma con el tercero y así sucesivamente.

Evidentemente, hay una estricta correspondencia entre los valores lógicos SI y NO con los dígitos binarios que son 1 y 0. A partir de este momento vamos asociar un dígito 1 a un SI y un dígito 0 a un NO.

De una manera general, el sumador debe resolver el problema siguiente: suponiendo que nos dan los valores de los dígitos A1, A2, B1 y B2 de la tabla

	A2	A1
+	B2	B1
C3	C2	C1

se deben calcular los valores de los dígitos C1, C2 y C3.

En nuestro programa las variables A1, A2, B1, B2, C1, C2 y C3 se simularán con variables lógicas.

El mecanismo de suma se describe para los tres dígitos del resultado de una manera distinta.

El primero de ellos, se obtiene al considerar la suma de los dígitos A1 y B1 únicamente. El segundo es más complicado pues hay que considerar los dos dígitos A2 y B2 y además recordar si se lleva o no una de la suma de los dígitos anteriores. Finalmente el tercer dígito, corresponde a colocar el dígito que memoriza si se lleva o no un dígito arrastrado de la suma de los segundos dígitos.

Debe observar que estos tres casos, se dan siempre aunque se sumen números de más cifras. El mecanismo de la suma de los dígitos es igual al del segundo en nuestro ejemplo, excepto, como en nuestro caso también, el primero y el último.

Lo que parece claro es que deberemos tener una variable dispuesta para controlar si llevamos o no llevamos una.

6.3.2 Diseño

En una primera parte deberemos preocuparnos por la entrada de los datos. Es necesario entrar los cuatro dígitos A1, A2, B1 y B2.

Esta entrada la haremos controlando que realmente sea un cero y un 1 lo que se entra. De esta manera, la variable numérica será a su vez lógica ya que sus únicos valores posibles serán el 0 y el 1, valores asociados al NO y el SI en el ZX-Spectrum.

Una vez recogidos estos datos se presentarán en pantalla en la forma siguiente:

```
          1         2         3
 12345678901234567890123456789012
1
2               XX
3             + XX
4            ------
5               xxx
6
```

Los números de la primera fila representan las columnas y los de la izquierda de cada fila representan las filas en pantalla.

Una vez entrados estos números se realizará el cálculo de la suma, se dispondrá en pantalla y se pedirá para realizar otra suma.

Si el primer dígito que entramos es −1, quiere decir que deseamos finalizar el programa.

Un esquema del programa es el siguiente:

— Borrar pantalla

— Presentar el fondo de pantalla

— Entrar el primer dígito (a2)

— Si es menos 1 finalizar

— Borrar los dígitos anteriores

— Colocarlo en pantalla

— Entrar el segundo dígito (a1)

— Colocarlo en pantalla

— Entrar el primer dígito del segundo (b2)

— Colocarlo en pantalla

— Entrar el segundo dígito del segundo (b1)

— Colocarlo en pantalla

— Efectuar la suma

— Colocarla en pantalla

— Volver a entrar el primer dígito

La parte más difícil es el paso de efectuar la suma; por ello, de momento no efectuaremos la suma y escribiremos esta parte de entrada de datos. Una vez probada pasaremos a considerar cómo se realiza la suma.

Con el fin de dejar los bloques separados; reservaremos el bloque de numeración de la línea 2000 a 3000 para el sumador propiamente dicho.

La traducción de estos pasos a código BASIC es la siguiente:

```
  10 CLS
  20 PRINT AT 3,12;"+";AT 4,12;"------";
1000 INPUT "A2:";a2
1010 IF a2= -1 THEN GOTO 9000
1020 IF a2<>0 AND a2<>1 THEN GOTO 1000
1030 PRINT AT 2,15;"  ";AT 3,15;"  "; AT 5,14;"   ";
1040 PRINT AT 2,15;a2;
1050 INPUT "A1:";a1
1060 IF a1<>0 AND a1<>1 THEN GOTO 1050
1070 PRINT AT 2,16;a1;
1080 INPUT "B2:";b2
1090 IF b2<>0 AND b2<>1 THEN GOTO 1080
1100 PRINT AT 3,15;b2;
1110 INPUT "B1:";b1
1120 IF b1<>0 AND b1<>1 THEN GOTO 1110
1130 PRINT AT 3,16;b1;
2000 REM Se efectua la suma
2010 LET c1=0: LET c2=0: LET c3=0
3000 PRINT AT 5,14;c3;c2;c1;
3010 GOTO 1000
9000 CLS
```

Observe que se utiliza frecuentemente la función AT de posicionado de cursor para dirigir la escritura en la pantalla de una manera controlada. Observe también que detrás de los PRINT está siempre un punto y coma, ya que como siempre escribimos dirigido hacia un lugar de la pantalla no hace falta saltar de línea.

Es mejor probar esta parte de programa antes que estudiar el sumador propiamente dicho.

De momento los resultados de esta suma van a ser cero siempre, ya que en la instrucción 2010, en lugar de sumar se colocan a cero los valores de c1, c2, c3.

Para probarlo, correctamente se debe controlar, en primer lugar que todo quede bien situado en la pantalla.

Después se debe intentar entrar números distintos de cero y uno, y controlar que efectivamente el programa los rechaza como entrada de datos.

Finalmente comprobar que el programa finaliza bien.

6.3.3 Sumador propiamente dicho

La técnica para sumar es un poco complicada, consideremos los dígitos primeros el A1 y B1. Si ambos son cero o ambos son uno el resultado es cero. Si uno es uno y el otro es cero, el resultado es 1. (De momento no se preocupe por la que llevamos).

Este comportamiento es precisamente el del operador XOR, da Si

cuando uno de los operandos es distinto y da NO cuando son iguales. En nuestro ordenador desafortunadamente no tenemos el XOR, pero ya sabemos por el capítulo de BASIC que es posible calcularlo mediante la expresión.

```
(NOT a1 AND b1) OR (a1 AND NOT b1)
```

por lo tanto, el resultado de esta expresión será el dígito que hay que colocar en c1.

La instrucción 2010 será:

```
2010 c1= (NOT a1 AND b1) OR (a1 AND NOT b1)
```

Ahora es el momento de considerar si llevamos una o no. La variable que utilizaremos para almacenar el arrastre es la c3, puesto que es la última que se calcula. En la suma del primer dígito sólo llevaremos una cuando a1 y b1 sean 1, es decir, SI. Por lo tanto, aplicando el operador AND entre ellos obtendremos si llevamos o no llevamos una. La operación

```
c3 = a1 AND b1
```

dará un 1 a c3 si llevamos una y un 0 si no llevamos ninguna.

Pasemos a considerar ahora los dígitos intermedios, de hecho, hay que sumar tres bits, el a2, el b2 y el c3 que es el que contiene si llevamos una o no.

Observe que aunque los tres dígitos sean 1, sólo podemos arrastrar como máximo 1, pues 1+1+1, en binario es 11 en binario y, por lo tanto, sólo llevamos una.

Para realizar la suma de estos tres dígitos sólo hay que repetir el caso anterior dos veces.

En efecto la suma de c3 y a2 será un primer valor de c2. Su cálculo se efectúa como

```
c2 = (NOT a2 AND c3) OR (a2 AND NOT c3)
```

este valor de c2 es provisional, pero en este momento responde al dígito sumado entre a2 y c3 (que es el que llevamos arrastrado).

Antes de hacer la otra suma hay que calcular si llevamos una o no para la siguiente operación. Observe que si llevamos una en esta parte de la operación, en la segunda parte no podrá llevar ninguna más. El hecho de

que en estos momentos ya lleve una es que el resultado en c2 es cero y, por lo tanto, al sumarle b2 no podrá sobrepasar el resultado de 1. El cálculo de ésta que llevamos la podemos almacenar en c3, con la operación

```
c3 = a2 AND c3
```

ya que la variable c3, que hasta ahora reflejaba la que llevábamos del dígito anterior, se ha utilizado al sumarlo con a2 y, por lo tanto, podemos alterar su valor.

Para finalizar la suma debemos sumar los dígitos b2 y c2 que nos quedan por sumar; sin embargo, el resultado lo tenemos que almacenar en c2, por lo tanto, antes de efectuar la suma propiamente dicha debemos saber si esta suma nos obligará a arrastrar un uno. Una vez realizada la suma puede haber cambiado el valor.

Esto sólo puede suceder si c3 es cero, recuerde que antes hemos menciona que el bit de arrastre sólo se puede modificar en una de las dos operaciones, y no en ambas a la vez. Por ello para saber si arrastramos o no una debemos realizar la instrucción siguiente:

```
IF NOT c3 THEN c3 = b2 AND c2
```

De hecho en los circuitos electrónicos no existe esta forma de IF, por lo tanto debemos sustituir este IF por una operación lógica. El valor de c3 debe ser uno si lo es c3 o si siendo cero c3 es uno b2 AND c2; luego la operación lógica indicada es el OR; por lo tanto, la instrucción anterior se debe escribir como

```
c3 = c3 OR ( b2 AND c2)
```

También en esta operación el valor de c3 proviene de la operación de cálculo de arrastre para la suma de a2 y la que llevamos antes; por lo tanto, la operación se puede escribir

```
c3 = ( a2 AND c3) or (b2 AND c2)
```

Una vez calculado el arrastre se puede ya calcular el valor definitivo del dígito c2 mediante la operación

```
c2 = (NOT b2 AND c2) OR (b2 AND NOT c2)
```

En este punto ya tenemos calculados los dígitos, pues c3 vale uno si se arrastra uno y vale cero si no se arrastra.

Esquematizado todo lo dicho:

Cálculo del primer dígito del resultado.

c1 = a1 XOR b1 : Calcula el dígito c1. Sólo es 1 si a1 y b1 son distintos.

c3 = a1 AND b1 : Calcula el arrastre del primer dígito. Sólo hay arrastre si los dos a1 y b1 valen 1.

Cálculo del segundo dígito del resultado.

c2 = c3 XOR a2 : Calcula el resultado del arrastre más el dígito a2. Se almacena provisionalmente en c2.

c3 = (c3 AND a2) or (b2 AND c2) : Calcula el arrastre de esta operación para el próximo dígito a sumar. Nunca pueden ser verdad simultáneamente los dos términos unidos por el OR.

c2 = b2 XOR c2 : Cálculo definitivo del segundo dígito.

Cálculo del tercer dígito del resultado.

c3 : Contiene ya el resultado; es uno si hay arrastre y cero si no lo hay.

En definitiva, el trozo de programa de sumador binario queda como sigue:

```
2010 LET c1 = (NOT a1 AND b1) OR (a1 AND NOT b1)
2020 LET c3 = a1 AND b1
2030 LET c2 = (NOT a2 AND c3) OR (a2 AND NOT c3)
2040 LET c3 = (a2 AND c3) OR (b2 AND c2)
2050 LET c2 = (NOT b2 AND c2) OR (b2 AND NOT c2)
```

6.3.4 Pruebas

Debido a que se trata de un sumador de sólo cuatro dígitos binarios a1, a2, b1 y b2 se puede comprobar su funcionamiento de una manera exhaustiva, ya que todas las combinaciones de cuatro bits son 16. Luego haciendo 16 pruebas sabremos si el sumador funciona correctamente.

Le ofrecemos a continuación estas 16 sumas para que pueda comprobar que el programa que ha realizado las reproduce.

```
  00    01    10    11
  00    00    00    00
 ----  ----  ----  ----
 000   001   010   011
  00    01    10    11
  01    01    01    01
 ----  ----  ----  ----
 001   010   011   100
  00    01    10    11
  10    10    10    10
 ----  ----  ----  ----
 010   011   100   101
  00    01    10    11
  11    11    11    11
 ----  ----  ----  ----
 011   100   101   110
```

6.4 PRACTICA 2. LA CONSTRUCCION DE FILTROS

Esta práctica consiste en realizar con un ejemplo la técnica del filtrado de información.

Siguiendo con el tema del capítulo de BASIC, vamos a diseñar un sistema que permita seleccionar distintos elementos en un fichero de objetos; elegimos coches, a partir de unas características que nos interesan en determinado momento.

Hemos mencionado el término fichero, pero lo vamos a utilizar en un sentido muy restringido, tómelo como una colección de objetos cuyas propiedades residen en la memoria del computador. Reservamos la definición rigurosa para el capítulo 17 del tomo V.

6.4.1 Definición del problema

Ya hemos dicho que vamos a considerar coches como objetos que queremos analizar, evidentemente no vamos a estudiar un coche en abstracto sino en función de unas propiedades que nos interesan.

Estas propiedades que vamos a seleccionar constituyen una representación del coche, es decir, unos aspectos del coche que nos interesan en estos momentos.

Estas propiedades son:

— Motor Diesel.

— Tracción Delantera.

— Frenos de Disco.

— 5 velocidades o marchas.

Estas propiedades son binarias en el sentido de que un determinado modelo de coche las tiene o no las tiene. Por lo tanto, nuestra representación del coche consistirá en cuatro valores SI o NO que se referirán a si posee o no posee la propiedad determinada.

Así, si el coche posee motor DIESEL, almacenaremos una S para esta propiedad y si no lo posee almacenaremos una N para esta propiedad. Lo mismo para la segunda, tercera y cuarta de las propiedades mencionadas.

Una imagen de nuestro fichero consistirá, por lo tanto, en una tabla como la que se muestra a continuación:

Modelo N.°	**MD**	**TD**	**FD**	**V5**
1	S	N	S	S
2	N	S	N	N
3	S	S	S	S
...				
n	S	N	S	S

Fíjese que éste es lo que más arriba denominamos fichero, la información sobre cada modelo está limitada a cuatro valores, por eso decimos que se trata de la representación de un coche, desde un punto de vista muy parcial que es el nuestro.

Decimos que además es muy especial porque para representarlo en la memoria del ordenador vamos a utilizar una cadena de caracteres S o N, normalmente no se utilizan las cadenas de caracteres para guardar ficheros.

Pero no lo vamos a hacer por filas, sino que vamos a tomar las columnas. Es decir, en el ordenador vamos a utilizar cuatro variables textuales, los nombres de las columnas d$, t$, f$ y v$, de tal manera que la tabla la dispondremos en memoria según la forma siguiente:

Modelo:	1	2	3 ...	n
Diesel d$:	S	N	S	S
Tracción t$:	N	S	S	N
Frenos f$:	S	N	S	S
Velocidades d$:	S	N	S	S

Es decir, en la primera posición de la variable textual d$, está el S si el modelo primero de coche tiene motor Diesel, y NO si no tiene motor Diesel, el carácter en la posición 3 de la variable textual d$ se refiere al modelo de coche número 3, el carácter en la posición 7 se refiere al modelo de coche número 7, y así sucesivamente.

La variable f$ contiene un S en la primera posición si el modelo de coche lleva frenos de disco y una N si no los lleva, lo mismo para el carácter que ocupa la posición 4 que se referirá al cuarto modelo de coche.

Esto se puede repetir para cada una de las variables textuales.

La operación de considerar el fichero en forma de columnas en lugar de filas se denomina inversión del fichero. Tenga en cuenta, que la forma en que los modelos son los números de fila es cómoda para rellenar, es decir, considerar cada modelo de coche y dar los valores de las cuatro propiedades; mientras que la que contiene los modelos en las columnas es cómoda para buscar y seleccionar la información.

El problema que vamos a plantear es el siguiente, deseamos realizar un programa que nos cree un fichero del estilo mencionado antes, para unos modelos de coche.

Cuando lo tengamos en memoria, vamos a realizar la selección de los modelos que cumplan con una pregunta que podamos hacer, por ejemplo, ¿Qué modelos llevan tracción delantera y no llevan motor Diesel?, de tal manera que el ordenador nos dé la lista de modelos que cumplen con esta condición lógica.

Lo vamos a hacer de tal manera que acepte cualquier pregunta que sea aceptable en BASIC con los operadores lógicos.

De la definición del problema ya se desprende que tendremos dos partes bien diferenciadas que son la entrada de datos de los modelos y la parte del programa que responda a las preguntas que le formulemos.

6.4.2 Entrada de datos

6.4.2.1 Diseño

La entrada de datos consiste en presentar una pantalla que tenga la forma siguiente:

```
          1         2         3
 12345678901234567890123456789012
1
2
3MODELO NUMERO: XX
4
5           Motor Diesel 1>x
6       Tracción delantera 2>x
7         Frenos de disco 3>x
8           5 Velocidades 4>x
9
```

Los números de la hilera superior indican las columnas de la pantalla y los números de la izquierda las filas de la pantalla. De esta figura deducimos que el rótulo del modelo debe estar en la fila 3 y columna 1. Mientras el número de modelo debe escribirse en la columna 16 de la fila 3. Por otra parte, los rótulos de las propiedades deben alinearse por la derecha en la columna 23 de las filas 5, 6, 7 y 8. Finalmente las propiedades deben presentarse en la columna 24 de las filas 5, 6, 7 y 8.

La numeración de los modelos será automática, es decir a medida que vamos entrando va dando el número al siguiente modelo de automóvil. Este número se almacenará en la variable n.

Utilizaremos cuatro variables d$, t$, f$ y v$ para almacenar cada uno de los atributos, que es otro nombre para las propiedades, es decir, hablar de propiedades y atributos es equivalente.

El primer paso a realizar es la presentación en pantalla, el esquema es el siguiente:

```
           CLS
           PRINT at 3,1;«MODELO NO:»
           a$= «Motor Diesel 1>»

repetir:   IF LEN(a$) <23 THEN a$=« »+a$
           PRINT at 5,1;a$
           a$= «Tracción delantera 2>»

repetir:   IF LEN(a$) <23 THEN a$=« »+a$: GOTO repetir
           PRINT at 6,1;a$
           ... y así para todos los atributos
```

(Observe que no utilizamos números de línea sino que únicamente utilizamos nombres, lo hacemos porque se trata sólo de un esquema y, por lo tanto, no estamos sometidos a los rigores sintácticos de la máquina. Este tipo de esquemas se utilizan mucho en el diseño de programas y debe considerarse normal en esta fase de construcción del programa. Luego cuando escribamos el programa completo ya traduciremos todo esto a números de líneas. De momento sólo queremos aclarar las ideas.)

El segundo paso consiste en realizar la pregunta de los atributos para cada modelo, el esquema de programa es el siguiente:

```
           n=0

otro:      n=n+1
           PRINT at 3,16;STR$(n)
           Preguntar por el valor de los atributos.
           INPUT «Hay más modelos»; a$
           PRINT at 5,24;« »;
           PRINT at 6,24;« »;
           PRINT at 7,24;« »;
           PRINT at 8,24;« »;
           IF a$<>«n» GOTO otro
           CLS
           finaliza la fase de entrada de datos
```

(se aplican las mismas observaciones hechas anteriormente respecto a la sintaxis de las líneas de programa).

En la primera parte se incrementa el valor de n y se presenta en pantalla, a continuación se debe entrar los valores de los atributos, después el programa nos pregunta si queremos entrar más modelos, después de la contestación se borran los atributos del modelo anterior y finalmente si la contestación es distinta de n(a$<>«n») se va a entrar otro modelo.

Vamos a desarrollar el diseño de la fase de entrada de los atributos.

La entrada se puede almacenar en una cadena de caracteres que tendrá de longitud máxima 4. El primer símbolo significará que si es una S, el

modelo que estamos entrando tiene el motor Diesel, si tiene una N es que no tiene el motor Diesel. La segunda posición significa lo mismo para la segunda propiedad. Lo mismo para la tercera y la cuarta propiedad. Para almacenar temporalmente estos atributos se utiliza la variable m$.

Se utiliza la variable p para contar los atributos entrados, es decir, vale 1 cuando entramos el atributo del Motor Diesel, 2 cuando entramos la tracción delantera y así sucesivamente.

El esquema de esta parte es:

```
          p=1:m$=»»
seguir:   IF p>4 GOTO salir
repetir:  INPUT «atributo «+STR$(p)+»:»;a$
          IF a$<>«s» AND a$<>«n» GOTO repetir
          m$=m$+a$:REM Se coloca el nuevo atributo en m$
          PRINT AT p+4,24;a$; : Se presenta en pantalla
          p=p+1
          GOTO seguir
salir:    Continuación del programa.
```

(le recordamos que es un esquema y que utilizamos un lenguaje no directamente aplicable al ordenador).

Después de esta fase de colección de resultados, es necesario invertir los valores entrados, es decir, colocar cada uno de los valores en las cadenas d$, t$, f$ y v$. (En la parte inicial de la entrada de datos las deberemos colocar al vacío.)

Para realizar esta parte utilizaremos el esquema siguiente:

```
d$=d$+m$(1 TO 1)
t$=t$+m$(2 TO 2)
f$=f$+m$(3 TO 3)
v$=v$+m$(4 TO 4)
```

La entrada para cada modelo consiste en ir alargando las cadenas d$, t$, f$ y v$ con los resultados entrados, de esta manera el carácter que está en la primera posición en las cadenas corresponde al modelo primero, el que está segundo al segundo modelo y así sucesivamente.

Como puede observar parece innecesaria la introducción de la variable n, pues no interviene para nada en la captura de datos. De todas maneras la variable n nos permite saber en que modelo estamos trabajando. Esto puede ser importante si estamos copiando de una lista pues cualquier distracción en la entrada puede hacernos perder el sitio donde trabajamos; la variable de contaje se encargará de informarnos donde estamos.

El esquema de la entrada de datos puede resumirse así:

```
          n=0: d$=«» : t$=«» : f$=«» : v$=«»
otro:     n=n+1
          PRINT AT 3,16;str$(n)
          p=1:m$=»»
```

```
seguir:   IF p>4 GOTO salir
repetir:  INPUT «atributo «+STR$(p)+»:»;a$
          IF a$<>«s» AND a$<>«n» GOTO repetir
          m$=m$+a$:REM Se coloca el nuevo atributo en m$
          PRINT AT p+4,24;a$; : Se presenta en pantalla
          p=p+1
          GOTO seguir
salir:    d$=d$+m$(1 TO 1)
          t$=t$+m$(2 TO 2)
          f$=f$+m$(3 TO 3)
          v$=v$+m$(4 TO 4)
          INPUT «Hay más modelos»; a$
          PRINT AT 5,24;« »;
          PRINT AT 5,24;« »;
          PRINT AT 7,24;« »;
          PRINT AT 8,24;« »;
          IF a$<>«n» GOTO otro
          CLS
          finaliza la fase de entrada de datos
```

Después de realizar este esquema, en las variables textuales d$, t$, f$ y v$ nos quedan dispuestos los atributos de los modelos que deseamos.

Antes de pasar a la discusión de la segunda parte vamos a codificar y probar esta parte.

6.4.2.2 Escritura del programa

A partir de los esquemas anteriores es fácil escribir el programa ya codificado para la máquina. Ud. debe intentar hacerlo sin consultar la versión que le ofrecemos. Una vez tenga su versión compárela con la nuestra para observar las diferencias y pase a la parte de pruebas, para ver si su versión es correcta. En todo caso, cuando haya experimentado con su versión, teclee la nuestra y siga con las pruebas.

```
 10 CLS
 20 PRINT AT 3,1;"MODELO NO:";
 30 LET a$="Motor Diesel 1)"
 40 IF LEN(a$)<23 THEN LET a$=" "+a$: GOTO 40
 50 PRINT AT 5,1;a$;
 60 LET a$="Traccion Delantera 2)"
 70 IF LEN(a$)<23 THEN LET a$=" "+a$: GOTO 70
 80 PRINT AT 6,1;a$;
 90 LET a$="Frenos de disco 3)"
100 IF LEN(a$)<23 THEN LET a$=" "+a$: GOTO 100
110 PRINT AT 7,1;a$;
120 LET a$="5 Velocidades 4)"
130 IF LEN(a$)<23 THEN LET a$=" "+a$: GOTO 130
140 PRINT AT 8,1;a$;
```

Cuando haya tecleado hasta este punto puede hacer un RUN y corregir los defectos que encuentre.

Continuemos ahora con la parte de entrada de datos, empezaremos en 1000 para que quede claramente separada de esta parte, en el interior de este nuevo bloque haremos un salto a 1500 para indicar la parte de entrada de los atributos y un salto a 1900 para acabar con la pregunta de si queremos continuar.

```
1000 REM Parte de la entrada de datos
1010 LET d$="": LET t$=""
1020 LET f$="": LET v$=""
1030 LET n=0
1040 LET n=n+1
1050 PRINT AT 3,16;STR$(n);
1500 REM Parte de entrada propiamente dicha
1510 LET p=1: LET m$=""
1520 IF p>4 THEN GOTO 1600
1530 INPUT "Atributo "+STR$(p)+":";a$
1540 IF a$<>"s" AND a$<>"n" THEN GOTO 1530
1550 LET m$=m$+a$
1560 PRINT AT p+4,24;a$
1570 LET p=p+1: GOTO 1520
1600 LET d$=d$+m$(1 TO 1)
1610 LET t$=t$+m$(2 TO 2)
1620 LET f$=f$+m$(3 TO 3)
1630 LET v$=v$+m$(4 TO 4)
1900 INPUT "Hay mas modelos:";a$
1910 PRINT AT 5,24;" ";AT 6,24;" ";
1920 PRINT AT 7,24;" ";AT 8,24;" ";
1930 IF a$<>"n" THEN GOTO 1040
1940 CLS: REM Finaliza la parte de entrada de datos
```

Para facilitar las pruebas de esta parte vamos a añadir que cuando finalicemos el programa nos escriba provisionalmente las variables n, d$, t$, f$ y v$, a fin de controlar que todo va bien en esta primera parte.

Las instrucciones serían las siguientes:

```
2000 PRINT n
2010 PRINT d$
2020 PRINT t$
2030 PRINT f$
2040 PRINT v$
```

6.4.2.3 Pruebas

Empezaremos las pruebas con un fichero pequeño, por ejemplo, de cuatro modelos de coche.

La tabla de este fichero puede ser la siguiente:

Modelo	Diesel	Tracción	Frenos	Velocidades
1	S	S	N	N
2	N	S	N	N
3	N	N	S	S
4	S	N	N	S

Después de ejecutar el programa tal como lo tenemos ahora debe aparecer en pantalla al finalizar la entrada las siguientes cadenas de caracteres:

4
SNNS
SSNN
NNSN
NNSS

Además de probar si intenta introducir algún atributo que no es *s* o *n* el programa no sigue adelante en este atributo hasta que la entrada sea correcta.

Debe comprobar además que el mecanismo de seguir entrando funciona correctamente.

6.4.3 Selección de información

Una vez tenemos el programa que nos carga los datos en memoria y nos invierte el fichero de entrada proseguimos con la fase de recuperación de información.

Recuerde que el objetivo de esta segunda parte consiste en recuperar la información de una manera selectiva, es decir, que cumpla una serie de condiciones que le manifestamos antes de proceder a la recuperación.

6.4.3.1 Diseño

El mecanismo básico consiste en formular una pregunta, realizar el cálculo de la información que cumple con la pregunta y formular otra vez la pregunta. Cuando, por ejemplo, la pregunta sea vacía se finaliza el programa.

El esquema es el siguiente:

```
otra        INPUT «Que pregunta desea formular:»;p$
            IF p$<>«» GOTO finalizar
            Mecanismo de contestación a la pregunta.
            GOTO otra

finalizar:  CLS
            El programa está finalizado.
```

Como puede ver el mecanismo global es muy sencillo, pasemos ahora a estudiar el mecanismo de contestación de la pregunta.

Antes, de todos modos, es necesario definir de qué manera haremos las preguntas. Utilizaremos cuatro variables lógicas, m, t, f y v. Estas variables lógicas tomarán el valor SI cuando el atributo del modelo considerado es s y el valor NO cuando la propiedad del modelo contiene una n. No debe confundir la utilización de las variables con el valor de los atributos. Debemos utilizar variables pues para cada modelo de coche el valor de la variable es distinto.

La inicial de la variable lógica responde a la idea del atributo, así d significa diesel, la t significa tracción, la f significa frenos y la v significa velocidades.

La manera de escribir la pregunta consiste en mezclar estas cuatro variables con los operadores NOT, AND y OR de manera que se forme una expresión lógica, válida en BASIC. Aprovecharemos las propiedades de la función VAL, que permite calcular expresiones contenidas en la variable textual que tiene como argumento.

Veamos algunos ejemplos de planteamiento de preguntas, imagine que desea formular la pregunta

¿Con motor diesel y sin 5 velocidades?

la traducción para formular la pregunta es

d AND NOT v,

en efecto, d significa que tenemos en cuenta el motor Diesel y será cierta cuando el coche tenga motor Diesel; v significa que cuando tenga 5 velocidades será cierta, como lo que queremos es sin 5 velocidades debemos negar esta variable. Finalmente en AND traduce la conjunción lógica y.

Si se aplica a la tabla que hemos entrado anteriormente, el resultado debe ser el modelo 1, ya que es el único que tiene motor Diesel, sólo lo tienen el 1 y el 4, y de estos dos sólo el 1 no tiene 5 velocidades.

Veamos un segundo ejemplo, queremos plantear la pregunta

¿Con tracción delantera o frenos de disco?

la traducción es:

t OR f,

en efecto, t será cierta cuando el modelo tenga tracción delantera, y f será cierta cuando los frenos sean de disco. El OR tiene en cuenta la partícula *o* de unión lógica que hacemos en el lenguaje ordinario.

Para que sea seleccionado un modelo bastará que posea al menos una propiedad de las dos mencionadas; en nuestro caso, el resultado es 1, 2, y 3. El único modelo que tiene las dos propiedades a NO es el 4.

Vistos estos dos ejemplos, vamos a desarrollar el esquema del programa:

```
CLS
PRINT p$
p=0
```

```
seguir:    p=p+1 : if p>n GOTO fin
           d = d$(p TO p) = «s»
           t = t$(p TO p) = «s»
           f = f$(p TO p) = «s»
           v = v$(p TO p) = «s»
           IF NOT VAL (p$) GOTO seguir
               El modelo cumple con la expresión
               Se imprime el resultado
               PRINT p;» »;
               GOTO seguir
fin:       se ha acabado la contestación a esta pregunta
```

Como se puede apreciar, este mecanismo consiste en repasar cada uno de los modelos de coche, el modelo que se está considerando es el que lleva el número p.

Antes de iniciar este recorrido se deja limpia la pantalla y se imprime la pregunta realizada.

Luego para cada coche se evalúan las variables m, t, f y v mediante la comparación a *s* del carácter que tiene el orden p dentro de la cadena de caracteres del correspondiente atributo.

A continuación se evalúa la expresión con la función VAL si el resultado es falso envía al programa a considerar otro caso, por eso se coloca el NOT delante de la evaluación de la función VAL.

Si el caso ha sido seleccionado se imprime el valor de p pues es el del modelo de coche que estamos analizando en aquel momento.

Finalmente se envía a que considere el modelo siguiente.

Cuando se alcanze un valor de p superior al número de modelos que contiene nuestro fichero se finaliza con el tratamiento a la pregunta, y se pasa a formular una nueva.

6.4.3.2 Escritura del programa

A la vista de este esquema debe intentar realizar Ud. mismo un programa que corresponda con este diseño, antes de consultar la versión que le ofrecemos.

La traducción es bastante directa y simple, el bloque más externo, el que se refiere a la pregunta lo empezaremos en 2000, y el bloque más interno de contestación a la pregunta lo empezaremos en 2500.

Antes de teclear nada, es necesario borrar las líneas 2000, 2010, 2020, 2030 y 2040 que nos han servido provisionalmente para comprobar los resultados de la parte de entrada de datos.

El listado de instrucciones es el siguiente:

```
2000 INPUT "Que pregunta desea formular:";p$
2010 IF p$="" THEN GOTO 3000
2500 CLS : PRINT p$ : LET p=0
2510 LET p = p+1: IF p>n THEN GOTO 2900
2520 LET d = d$(p TO p) = "s"
2530 LET t = t$(p TO p) = "s"
```

```
2540 LET f = f$(p TO p) = "s"
2550 LET v = v$(p to p) = "s"
2560 IF NOT VAL p$ THEN GOTO 2510
2570 PRINT p;" "; : GOTO 2510
2900 GOTO 2000
3000 CLS: REM Ha finalizado el programa
```

6.4.3.3 Pruebas

Las primeras pruebas que podemos realizar son las preguntas que hemos formulado cuando describíamos como se planteaban, es decir,

d AND NOT v

y

t OR f

A éstas les podemos añadir, las dos siguientes:

¿Motor Diesel y frenos de disco
o tracción delantera y 5 velocidades?

y

¿Motor de gasolina y tracción delantera o
frenos de disco?

Estas preguntas traducidas a nuestro esquema de preguntas son:

(d AND f) OR (t AND v)

y

NOT d AND (t OR v)

Los resultados de la primera es que no hay ningún modelo que la cumpla, mientras que el resultado de la segunda son los modelos 2 y 3.

6.4.4 Aplicación

Para finalizar con esta práctica vamos a cargar un fichero mayor para comprobar un poco todas las posibilidades.
Sea el fichero siguiente extraído de una fuente desconocida:

Modelo	Motor	Tracción	Frenos	Velocidades
1	n	s	s	n
2	s	s	n	n
3	n	s	s	n
4	n	s	s	s
5	s	s	s	s
6	n	n	n	n
7	s	n	n	n
8	n	n	n	s
9	s	s	n	n
10	n	n	n	n
11	n	s	s	n
12	n	n	n	s
13	s	s	n	s
14	n	n	s	s
15	n	n	s	n
16	s	n	s	s
17	s	n	s	n
18	n	n	n	s
19	n	s	s	s
20	n	s	n	n

Introduzca este fichero en el programa.

¡Sea cuidadoso no se equivoque al entrar los datos, los resultados pueden ser totalmente distintos!

Para comprobar que ha entrado correctamente el fichero, haga las cuatro preguntas siguientes:

d

t

f

v

En cada una de las preguntas le debe aparecer la lista de modelos que tienen una s en el atributo considerado. Así para d la lista debe ser

2,5,7,9,13,16,17

que son los modelos que tienen motor diesel.

Para las otras variables la lista es

t : 1,2,3,4,5,9,11,13,19,20

f : 1,3,4,5,11,14,15,16,17,19

v : 4,5,8,12,13,14,16,18,19

Formulemos las cuatro preguntas que hemos probado con el fichero pequeño.

A la pregunta

d AND NOT v

el resultado es

2,7,9,17

a la pregunta

t OR f

el resultado es

1,2,3,4,5,9,11,13,14,15,16,17,19,20

a la pregunta

(d AND f) OR (t AND v)

el resultado es

4,5,13,16,17,19

a la pregunta

NOT d AND (t OR v)

el resultado es

1,3,4,8,11,12,14,18,19,20

No borre el programa porque en las pruebas de evaluación lo va a necesitar para realizar las preguntas 11 a 15 de las pruebas prácticas.

Capítulo 7

ESQUEMAS DE CONTENIDO

Objetivos	
Función INKEY$	Visualización temporal Selección opciones Reconocimiento de teclas
Diseño de programas	Ecuación de segundo grado Actualización de precio Números pares Interés compuesto
Tablas de decisión	Clasificación notas Clasificación piezas
Práctica 1. Realización de una factura	
Práctica 2. Los filtros lógicos	

7.1 FUNCION INKEY$

La función INKEY$ se encuentra escrita en la tecla N. Por tanto, para utilizarla recordemos que hay que pasar primero a modo E. Su funcionamiento, en principio, es idéntico al estándar. Sin embargo, el teclado del ZX-SPECTRUM tiene la particularidad de que memoriza la tecla pulsada durante algunas fracciones de segundo. Para comprobarlo, escribamos el siguiente programa:

```
10 LET A$=INKEY$
20 IF A$="" THEN GOTO 10
30 PRINT "HA PULSADO:";A$
40 GOTO 10
```

El funcionamiento esperado de este programa sería que al pulsar una tecla cualquiera, por ejemplo, la A, visualizara en pantalla el mensaje «HA PULSADO:A». Entonces el control pasa de nuevo a la línea 10 y el programa queda bloqueado entre la 10 y la 20 hasta que pulsemos una nueva tecla. Sin embargo, en el ZX-SPECTRUM veremos que al pulsar una sola vez la letra A, el mensaje «HA PULSADO:A» puede aparecer repetido varias veces en pantalla (puede también que aparezca una sola vez). Esto se debe, como hemos mencionado, a que la letra A queda memorizada durante unos instantes. Entonces, después de escribir la primera vez el mensaje, pasa control a la línea 10, donde la función INKEY$ encuentra que todavía está memorizada la tecla A, actuando por tanto como si efectivamente la hubiéramos pulsado de nuevo. Este proceso se repite varias veces hasta que se desvanece la memorización.

La tecla pulsada queda memorizada algunos segundos

Como que este funcionamiento puede originar comportamientos extraños en algunos programas, habrá que utilizar un procedimiento para evitarlos. Está claro que la solución consistirá en impedir que el control pase de nuevo a la función INKEY$ hasta que no se haya desvanecido la tecla. Como sabemos, en el ZX-SPECTRUM, existe la instrucción PAUSE que nos ofrece precisamente la posibilidad de detener el programa durante un cierto tiempo.

Si la instrucción PAUSE tiene un valor suficientemente elevado, la memorización de la tecla se desvanecerá y entonces dejaremos paso a la función INKEY$. En lugar de efectuar algunas pruebas para hallar cual es este valor, utilizaremos la instrucción.

```
PAUSE 0
```

Con este valor, como ya estudiamos en su momento, la instrucción se espera indefinidamente hasta que se pulse una tecla. Para la instrucción PAUSE no tiene efecto la tecla memorizada, sino que necesita que se produzca una pulsación. Por consiguiente, en el ZX-SPECTRUM colocaremos siempre una instrucción PAUSE precediendo a la función INKEY$.

En el segundo programa del texto deberá modificar la línea 10 y el programa le quedará

```
10 PAUSE 0: LET A$=INKEY$
20 IF A$="" THEN GOTO 10
30 PRINT "TECLA=";A$
```

En el programa que sirve para determinar los códigos de las teclas especiales, introduciremos dos modificaciones respecto al BASIC estándard. La primera es la instrucción PAUSE en la línea 10. La segunda es cambiar la función ASC del BASIC estándar por la función CODE en la línea 30. Recordemos que ya se mencionó este cambio de nomenclatura en la sección dedicada a las funciones. El programa queda:

```
10 PAUSE 0: LET A$=INKEY$
20 IF A$="" THEN GOTO 10
30 PRINT "CODIGO=";CODE(A$)
```

7.1.1 Visualización temporal

En el programa que realizará en el apartado correspondiente a la visualización temporal en el capítulo dedicado al BASIC estándar, tenga la precaución de modificar la línea 60, añadiéndole la instrucción PAUSE. La línea quedará así:

```
60 PAUSE 0: LET A$=INKEY$
```

7.1.2 Selección opciones

La línea 50 del programa que realizará en el apartado de selección de opciones en la lección dedicada al BASIC estándard debe cambiarla así:

```
50 PAUSE 0: LET A$=INKEY$
```

Por otra parte, la línea 9010 la cambiaremos por el equivalente del END en el ZX-SPECTRUM, es decir, con un GOTO a una línea mayor que las existentes; por ejemplo:

7.1.1

```
9010 GOTO 9100
```

7.1.3 Reconocimiento de teclas

Al igual que en los demás programas, la línea 50 queda:

Siempre debe colocar un PAUSE Ø antes de la función INKEY$

```
50 PAUSE 0: LET A$=INKEY$
```

Cuando hayamos probado unas cuantas teclas de este programa, el ordenador nos pondrá el mensaje de «scroll?». En este momento no estamos bajo control de la función INKEY$, por tanto pulsaremos ENTER para que siga normalmente la ejecución del programa.



7.2 DISEÑO DE PROGRAMAS

7.2.1 Ecuación de segundo grado

El programa en BASIC correspondiente al ordinograma estudiado es el siguiente: (vaya siguiendo el ordinograma a medida que escribe el programa)

```
 10 INPUT "Entre el valor de A: ";A
 20 INPUT "Entre el valor de B: ";B
 30 INPUT "Entre el valor de C: ";C
 40 LET D=B*B-4*A*C
 50 IF D<0 THEN GOTO 100
 60 LET R1=(-B-SQR(D))/(2*A)
 70 LET R2=(-B+SQR(D))/(2*A)
 80 PRINT R1,R2
 90 STOP
100 PRINT "SOLUCION COMPLEJA"
```

El bloque de lectura se ha traducido por las instrucciones 10, 20 y 30 aunque se hubiera podido emplear una sola instrucción INPUT con las tres variables. El primer bloque de proceso corresponde a la línea 40. Observe que en esta línea se realiza el cálculo de lo que cae bajo la raíz cuadrada para saber si el valor será positivo o negativo. El resultado se almacena en D.

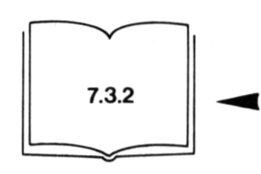
En la línea siguiente se encuentra la traducción del símbolo de decisión. Observemos que uno de los caminos lleva a la línea 100 y el otro sigue por la 50, según el valor almacenado en D sea positivo o negativo.

En las líneas 60 y 70 se calculan las raíces. En las operaciones que se realizan para hallarlas, se han colocado algunos paréntesis a fin de que se efectúen en el orden correcto.

Observe que, en lugar de repetir toda la fórmula, para calcular lo que cae bajo la raíz, se ha puesto la variable D, que ya estaba calculada en la línea 40. Por lo tanto, no hace falta repetir esta parte.

En el ordinograma no se dan los detalles de los formatos de escritura

Recordemos que en los ordinogramas no se dan detalles respecto al formato de escritura. En este caso hemos optado por la traducción más sencilla, pero nada impide modificar las líneas 80 y 100 para que escriban los resultados de otra forma. Por otra parte, este programa tiene dos puntos finales. Por tanto, uno de ellos debe quedar situado explícitamente (línea 90) mientras que el otro es automático ya que detrás de la línea 100 no vienen más instrucciones.



7.2.2 Actualización de precio

El programa que realiza este proceso es el siguiente: (siga el ordinograma)

```
10 INPUT "Tanto por ciento: ";T
20 INPUT "Precio: ";P
30 IF P=0 THEN STOP
40 LET R=P*(1+T/100)
50 PRINT R
60 GOTO 20
```

En este programa es obligatorio utilizar dos instrucciones INPUT separadas. El bloque de decisión está representado por la línea 30. Si P es cero, el programa finaliza directamente. En caso contrario, pasa a la fase de cálculo y escritura. La instrucción GOTO de la línea 60 equivale a la flecha que retrocede en el ordinograma. Aunque en el gráfico, el punto de llegada de esta flecha está situado entre los dos bloques de entrada, en BASIC esto no tiene sentido. Por ello, la instrucción GOTO va dirigida a la línea 20.



7.2.3 Números pares

Para determinar si un número es par, emplearemos el siguiente programa construido a partir del ordinograma del texto:

```
10 INPUT "NUMERO:";N
20 IF N<1 THEN GOTO 50
30 LET N=N-2
40 GOTO 20
50 IF N=0 THEN GOTO 80
60 PRINT "ES IMPAR"
70 STOP
80 PRINT "ES PAR"
```

En la línea 10, se extrae el número que se almacena en la variable N. Las líneas 20, 30 y 40 forman el bucle donde se realizan las restas. Cuando N es menor que 1 (línea 20) se pasa control a la línea 50 donde se determina si N vale cero o no. En caso afirmativo se escribe que es par (línea 80). En caso negativo se escribe que es impar (línea 60). Este programa tiene dos puntos finales. El primero está situado en la línea 70 y el segundo está situado de forma ímplicita después de la línea 80.

Como ya hemos mencionado, este programa sólo tiene interés para adquirir práctica en programación. Sin embargo, su eficiencia es muy reducida, como es fácil de comprobar. Si N es mayor de 100, el programa tarda bastante para averiguar si el número es par o no. Si realmente nos interesa determinar si un número es múltiplo de dos de una manera rápida y eficaz, emplearemos la función INT. El programa queda entonces:

```
10 INPUT "NUMERO:";N
20 IF N=2*INT(N/2) THEN GOTO 50
30 PRINT "IMPAR"
40 STOP
50 PRINT "PAR"
```

Cálculo con la función INT para averiguar si un número es par

El funcionamiento de la línea 20 es el siguiente. La función INT tiene el efecto de suprimir los decimales. Si dividimos N por 2, nos dará un resultado exacto si N es múltiplo de 2. Por tanto, la función INT no tendrá ningún efecto. Al multiplicar de nuevo por 2, recuperamos el valor inicial de N. El IF de la línea 20 precisamente pregunta esta igualdad. Por ejemplo, si N vale 4 entonces

2*INT(4/2) = 2*INT(2) = 2*2 = 4

es decir, que el resultado es 4 de nuevo. Por el contrario, si N no es múltiplo de 2, entonces la división dará un resultado con decimales, los cuales serán suprimidos por la función INT. Por tanto, al multiplicar por 2 se obtendrá un número distinto de N. Por ejemplo, si N vale 9 entonces

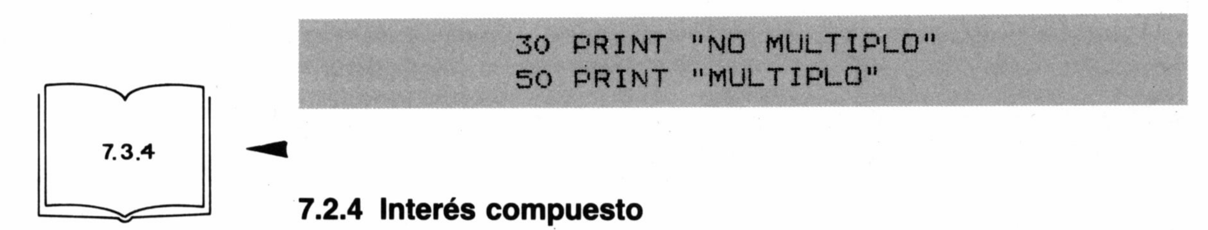
2*INT(9/2) = 2*INT(4.5) = 2*4 = 8

El valor obtenido (8) difiere del inicial (9).

Este programa determina de forma instantánea si un número es par, sea cual sea el tamaño de N. Si en lugar de averiguar si el número es múltiplo de 2, queremos determinar si es múltiplo de otro número, se cambia el 2 de la línea 20 por el número apropiado. Por ejemplo, para los múltiplos de 3 escribimos

```
20 IF N=3*INT(N/3) THEN GOTO 50
```

Lógicamente, las instrucciones de escritura deberán cambiarse también, quedando:



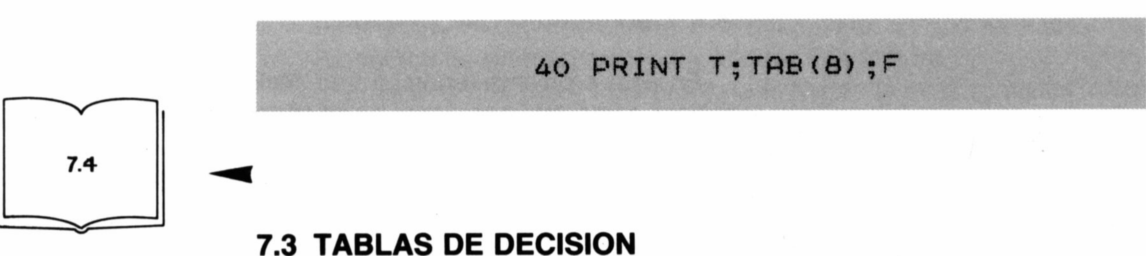

```
30 PRINT "NO MULTIPLO"
50 PRINT "MULTIPLO"
```

7.2.4 Interés compuesto

El listado del programa es el siguiente

```
10 INPUT C,R
20 LET T=1
30 LET F=C*(1+R/100)^T
40 PRINT T,F
50 LET T=T+1
60 IF T<=5 THEN GOTO 30
```

El valor del capital y del rédito se introducen en la línea 10. A continuación, se define la variable T que contendrá el número de años. En la línea 30 se efectúa el cálculo del capital final. En la expresión numérica empleada en esta instrucción se ha tenido en cuenta el orden de prioridad de los operadores. Una vez calculado, se escribe T y F, repitiéndose el proceso para un nuevo valor de T. El bucle está formado por las líneas comprendidas entre la 30 y la 60. En los ordinogramas no se especifican los detalles sobre la manera en que quedará escrito el resultado final. Es en la fase de codificación cuando se decide este punto. Por ejemplo, si deseamos encolumnar los resultados de forma precisa, emplearemos la función TAB. Entonces, la línea 40 quedaría:




```
40 PRINT T;TAB(8);F
```

7.3 TABLAS DE DECISION

7.3.1 Clasificación notas

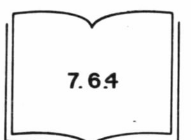
El programa que corresponde a la tabla de decisión de la figura 18 de la lección BASIC estándar, utilizada para clasificar las notas es el siguiente:

```
10 INPUT N
20 IF N>=3 THEN GOTO 50
30   PRINT "MUY DEFICIENTE"
```

```
 40    GOTO 150
 50 IF N>=5 THEN GOTO 80
 60    PRINT "DEFICIENTE"
 70    GOTO 150
 80 IF N>=6 THEN GOTO 110
 90    PRINT "SUFICIENTE"
100    GOTO 150
110 IF N>=9 THEN GOTO 140
120    PRINT "BIEN"
130    GOTO 150
140 PRINT "EXCELENTE"
150 STOP
```

El programa se construye con la técnica de alternativas múltiples

El programa se construye de tal forma que una instrucción IF se encadene con otra. Estas instrucciones tienen la pregunta escrita en forma inversa que en la tabla de decisión, de modo que si el resultado es cierto (falso en la tabla), se pasa control al siguiente IF. Si por el contrario, la condición es falsa (cierta en la tabla) se efectúa la acción o tratamimento asociado. Seguidamente se coloca siempre un GOTO para pasar control directamente a la última línea. En este ejemplo, todas las condiciones se basan sobre una misma variable (la nota). Por consiguiente, la entrada de datos del programa estará formada por una instrucción INPUT con la variable N (línea 10). Escribiendo un valor de la nota comprendido entre 0 y 10, el programa nos dará un mensaje indicando la clasificación de la nota según las reglas expuestas en la tabla de decisión.



7.3.2 Clasificación de piezas

En este problema tenemos tres acciones distintas a tomar, que son: clasificar como primera clase, clasificar como segunda clase y desechar. Las cuatro condiciones del problema incluyen una pregunta doble. Por ejemplo, que el diámetro de la pieza sea *mayor o igual que 99* y que sea *menor o igual que 100*. Para efectuar estas dos preguntas en BASIC utilizamos los operadores lógicos o booleanos, de modo que con una sola instrucción IF realicemos las dos preguntas a la vez. El listado es el siguiente:

```
10 INPUT "Diametro y longitud:";D,L
20 IF 500<=L AND L<=501 THEN GOTO 50
30   PRINT "DESECHAR"
40   GOTO 120
50 IF D<99 OR D>100 THEN GOTO 80
60   PRINT "PRIMERA CLASE"
70   GOTO 120
80 IF D<98 OR D>101 THEN GOTO 110
```

```
  90   PRINT "SEGUNDA CLASE"
 100   GOTO 120
 110 PRINT "DESECHAR"
 120 STOP
```

Decisiones con dos variables

Las cuatro condiciones del problema se basan sobre dos variables, el diámetro y la longitud. La entrada de datos estará formada pues, por una instrucción INPUT donde se lean los valores de D y L, tal como se ve en la línea 10 del listado. A continuación realizaremos la primera pregunta sobre la longitud de la pieza. Si la longitud es incorrecta, se escribe el mensaje correspondiente y nos dirigimos directamente al final, puesto que ya no tiene sentido seguir el proceso. De hecho, este modo de operar se corresponde exactamente con el significado de los casilleros que tienen el guión. Como ya sabemos, el guión indica que aquella condición no es significativa en aquel momento, tal como ocurre en este caso.

Si la longitud es correcta, tenemos todavía tres casos que se deciden mediante dos instrucciones IF encadenadas (las líneas 50 y 80). Al ejecutar el programa nos pedirá el valor del diámetro y de la longitud. Le entraremos dos números cualesquiera y el programa nos escribirá un mensaje indicando cómo se ha clasificado la pieza.

7.6.5

7.4 PRACTICA 1. REALIZACION DE UNA FACTURA

Vamos a realizar el ordinograma de la práctica primera del capítulo de prácticas de este tomo.

Para que no tenga que consultar le resumimos en qué consiste.

Se trataba de un programa para realizar una factura.

El programa nos pide como datos, la fecha de la factura, varias líneas de dirección del destinatario, una serie de artículos, de los cuales hay que especificar el nombre, la cantidad pedida, y el precio unitario. Cuando el nombre del artículo era la cadena vacía el programa pasa a pedirnos el impuesto que hay que aplicar.

Finalmente calcula los totales y el programa se detiene.

El programa se lo reproducimos a continuación para que concrete las ideas anteriores.

```
1000 REM ------------ Entrada de la fecha
1010   INPUT "Fecha: ";f$
1020   CLS : PRINT TAB (9); f$
2000 REM ----------- Entrada de las lineas de
                     direccion
2010   INPUT "Linea de direccion: ";d$
2020   PRINT TAB(15);d$
2030   IF d$<>"" THEN GOTO 2010
```

```
2040   PRINT : PRINT
3000 REM ----------- Entrada de los articulos
3010   LET t=0
3020   INPUT "Articulo: ",n$
3030   IF n$="" THEN GOTO 4000
3040   INPUT "Cantidad: ";q
3050   INPUT "Precio unitario: ";p
3060   LET i=p*q : LET t=t+i : REM Calculo
                               importe acumu
                               lacion del
                               total
3070   LET l=LEN n$
3080   LET b$="..............."
3090   LET b$(1 TO l)=n$
3100   PRINT b$;" ";
3110   LET col=1
3120   IF col=1 THEN LET lt=4 : LET num=q
3130   IF col=2 THEN LET lt=4 : LET num=p
3140   IF col=3 THEN LET lt=5 : LET num=i
3150   LET a$=STR$(num) : LET nb=lt-LEN (a$)
3160   IF nb>0 THEN LET a$=" "+a$ :
       LET nb=nb-1:GOTO 3160
3170   PRINT a$;" ";
3180   LET col=col+1
3190   IF col<=3 THEN GOTO 3120
3200   PRINT
4000 REM ---------------Escritura de totales
4010   PRINT TAB(26);"----"
4020   PRINT TAB(19);"Total..";
4030   LET a$ = STR$(t)
4040   LET l = LEN(a$)-1
4050   LET b$= "     "
4060   LET b$(5-l TO 5)=a$
4070   PRINT b$;" "
4080   PRINT : LET i=INT (t*tpc/100+0.5)
4090   PRINT TAB(15);"Impuesto ";STR$(tpc);"";
4100   LET a$=STR$(i)
4110   LET l=LEN(a$)-1
4120   LET b$="     "
4130   LET b$(5-l TO 5)=a$
4140   PRINT b$;" "
4150   PRINT TAB(26);"-----"
4160   LET t=t+i
4170   LET a$=STR$(t)
4180   LET l=LEN(a$)-1
4190   LET b$="     "
4200   LET b$(5-l TO 5)=a$
4210   PRINT TAB(26);b$;" "
4220   PRINT TAB(26);"====="
```

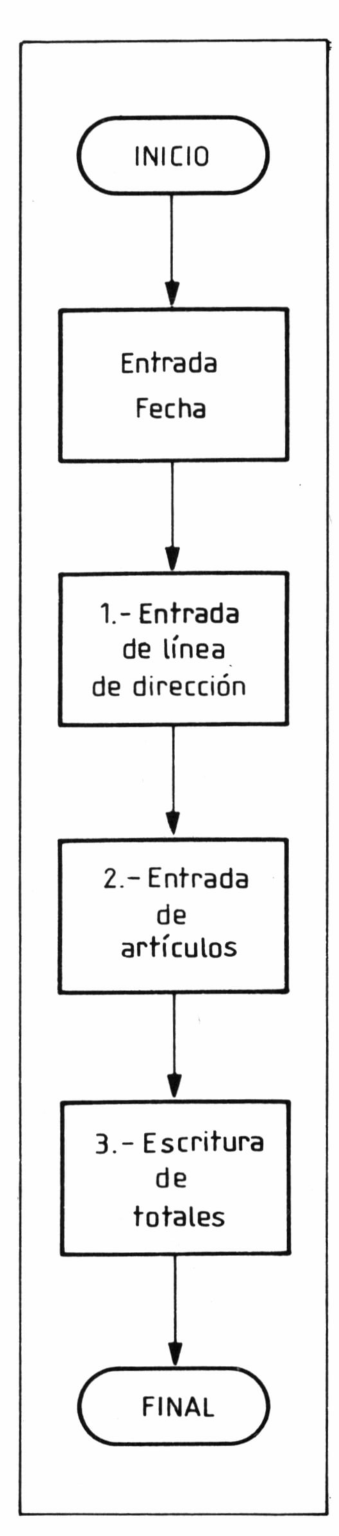


Figura 1: Ordinograma de primer nivel de la realización de facturas.

El objetivo de esta práctica es realizar los ordinogramas correspondientes al diseño jerárquico.

De alguna manera, parece ilógico hacer el ordinograma de un programa que ya está hecho y funciona. Recuerde, sin embargo, que el ordinograma sirve también para documentar el programa.

Sin más verá que debe hacer un esfuerzo notable para recordar cada cosa del programa, seguramente está tentado en consultar el capítulo 5.

La documentación, por otra parte debe reflejar el programa correctamente, por lo tanto, hay que revisarla una vez el programa está acabado.

Dentro del ambiente de este curso, también es interesante hacer ejercicios de traducción a la inversa, es decir, lo normal es traducir de ordinogramas a lenguajes de programación, pero como ejercicio también es muy útil hacer el paso inverso. Como en el aprendizaje del inglés, hacemos traducciones del castellano al inglés y del inglés al castellano.

Ya cuando realizamos el programa utilizamos una numeración generosa para destacar bien los bloques, introdujimos los comentarios oportunos para clarificar la estructura.

Este puede ser nuestro primer ordinograma, vea la figura 1.

Observe que cada bloque se ha numerado; se ha hecho para que luego al desarrollar cada uno de ellos por separado sepamos a que bloque corresponde.

El bloque de entrada de la fecha no tiene mayores problemas. Se puede considerar finalizado a este nivel. El hecho de que no lleve numeración indica que se ha terminado el desarrollo del bloque.

Ciertamente se puede usar un bloque de entrada y salida, en lugar de un bloque de proceso en general, tenga en cuenta que ya hemos dicho en la lección del BASIC estándar que debemos fijarnos en la semántica no en la sintaxis; no debemos ser esclavos de la sintaxis. Esto ya lo exige el lenguaje de programación. En consecuencia dejaremos la entrada de la fecha como un proceso en general.

El bloque de entrada de las líneas de dirección ya merece un desarrollo más detallado, para poner de manifiesto la estructura de control que contiene. Tenga en cuenta que se van entrando líneas de dirección hasta que hay una que es vacía.

La figura 2 muestra de forma más detallada este bloque. El ordinograma se inicia con un conector de página, realmente ésta no es la utilización correcta del símbolo, pero volvemos a insistir en que no hay una sintaxis rigurosa. De esta manera sabemos que el ordinograma representa el desarrollo de un bloque que se ha mencionado anteriormente. También el ordinograma finaliza con un símbolo de conexión entre páginas para indicar que se vuelve al final del bloque de proceso numerado, en el ordinograma de nivel superior.

A continuación se realiza un bloque de proceso de entrada y a continuación un bloque de proceso de salida. Finalmente un bloque de toma de decisiones; si la línea entrada no es vacía, se recicla el proceso; en caso contrario, se colocan los espacios de separación mediante la impresión de dos líneas en blanco, además de la línea vacía que hemos entrado.

El bloque de entrada de artículos debe desarrollarse más, lo haremos en tres etapas.

La figura 3 muestra sobretodo la estructura de control; se inicia con el

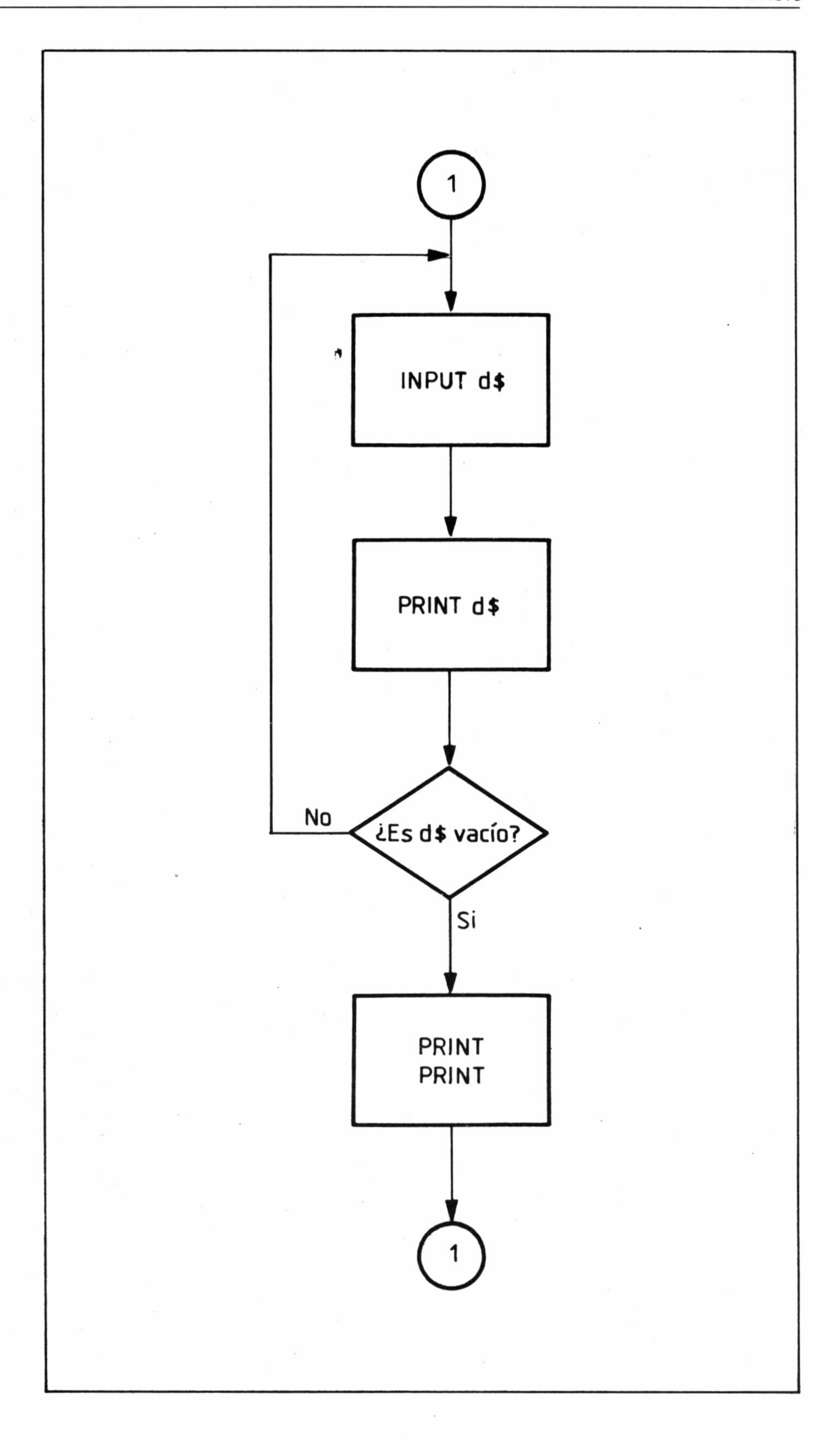


Figura 2: Ordinograma de desarrollo del bloque «Entrada de línea de la dirección».

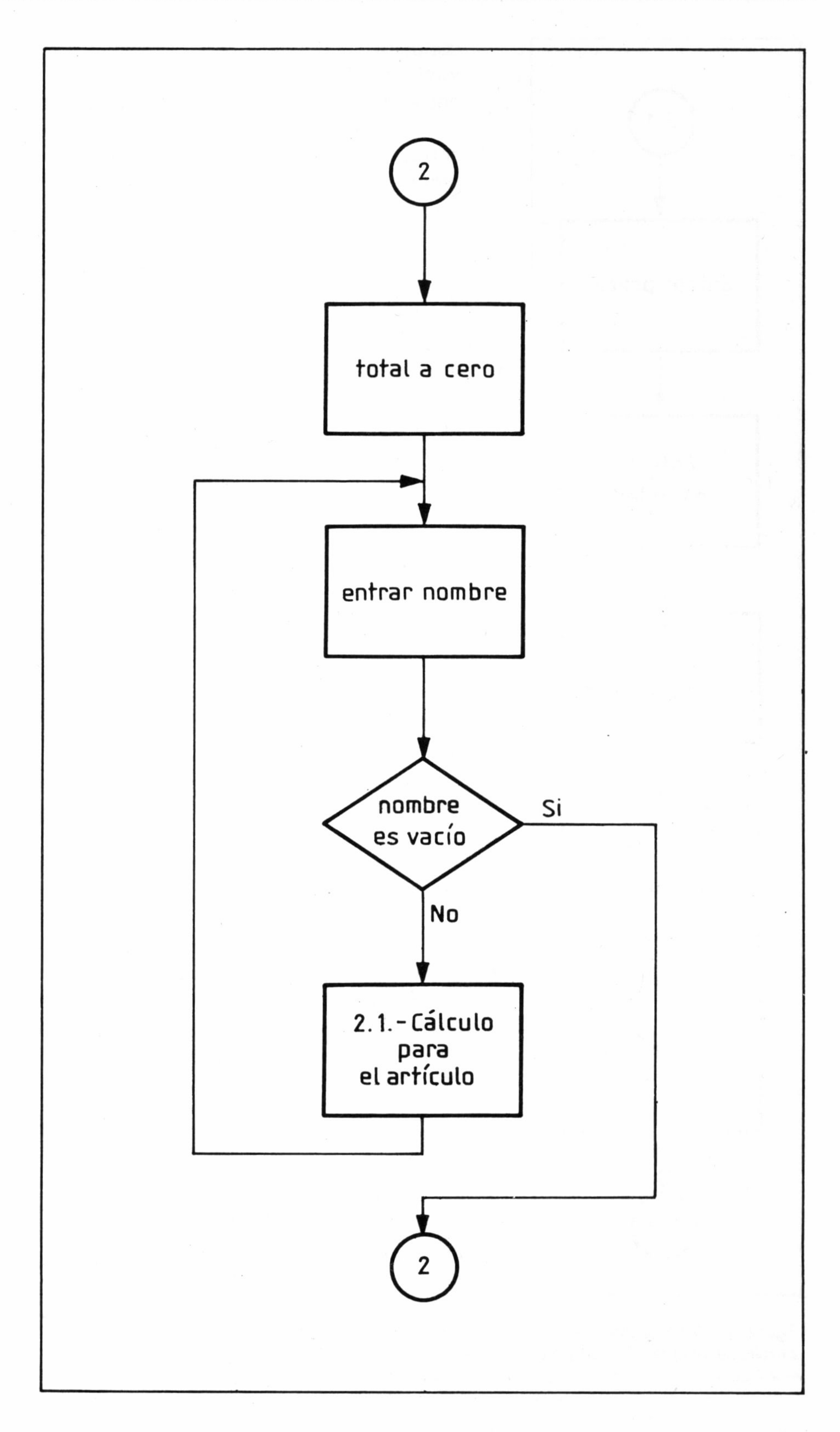


Figura 3: Ordinograma de desarrollo del bloque de «Entrada de artículos».

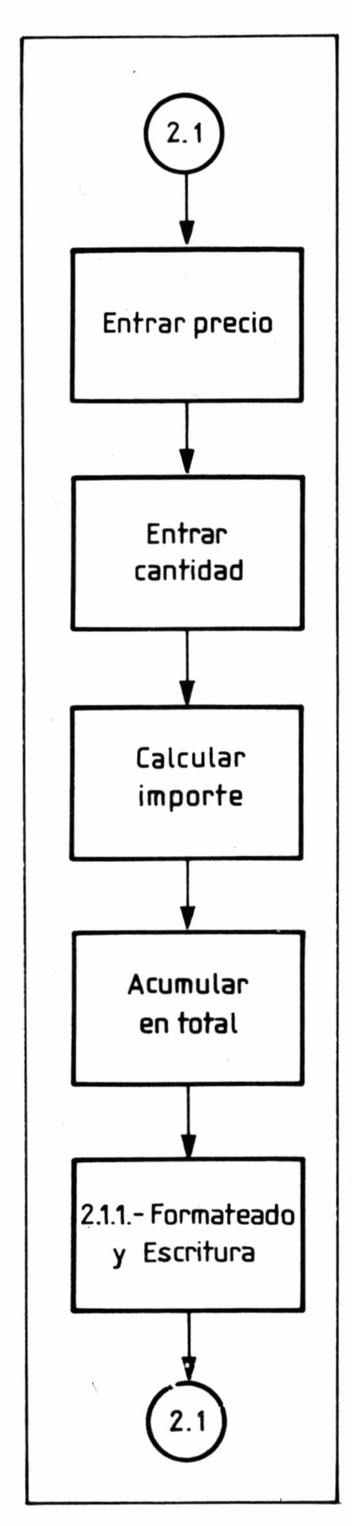


Figura 4: Ordinograma de desarrollo del bloque «Cálculo para el artículo».

conector para indicar de qué bloque proviene; a continuación se coloca el total de la factura a cero, la entrada del nombre del artículo es el bloque de entrada salida, se toma la decisión de calcular si el nombre no es vacío.

Se entra ahora en el bloque de cálculo del artículo. Este va numerado con 2.1 pues indica que hay un ordinograma posterior que explicita más el proceso. Una vez finalizado este cálculo se recicla a preguntar otro nombre de artículo.

Los cálculos que se deben realizar para cada artículo se muestran en la figura 4. Como puede Ud. apreciar se inicia con el conector de página 2.1. Indica que es el subordinograma 1 del bloque 2 del ordinograma de nivel 0. Este ordinograma contiene bloques de proceso que especifican las entradas adicionales y los cálculos a realizar. Sin embargo aún nos queda el bloque del formateado que está por decidir. Ya se ha mencionado que en general estos detalles no se incluyen en el ordinograma, pero justamente en el caso de este ejemplo es un poco más complicado que lo normal. Por lo tanto, realizaremos otro ordinograma que muestre estos detalles. Además este bloque va numerado con 2.1.1, esto nos indica que existe un ordinograma que explica con más detalle este bloque.

Finalmente el ordinograma acaba con el conector 2.1 que nos indica donde debemos volver en el ordinograma de nivel superior.

En el ordinograma de la figura 5, después del conector, aparece una primera parte que consiste en situar el nombre sobre un fondo de puntos para trazar mejor las líneas cuando se consulta la factura. Se imprime este nombre y se pasa a escribir las columnas; en una primera parte, una secuencia de IF asigna, según la columna que estamos escribiendo, la longitud del número y el valor del número a las variables lt y num que son las que se utilizarán para escribir.

Después aparece un bloque de cálculo que obtiene el número de blancos que hay que añadir a la derecha del número para que tenga la longitud especificada.

Una toma de decisión y un bloque va controlando que se añadan los blancos necesarios, en el programa esto corresponde únicamente a la línea 3160.

Finalmente se imprime la columna, se incrementa en uno la columna y si es inferior a tres se vuelve a escribir la siguiente columna.

Resta desarrollar el último bloque de la figura 1, que corresponde a la escritura de totales, que no presenta problema alguno ya que es una secuencia de instrucciones que prácticamente se pueden escribir (o describir) por una lista de acciones como la siguiente:

====Proceso 3

Imprimir una raya.

Imprimir la palabra «Total».

Imprimir el número ajustado a la derecha.

Entrar el impuesto.

Calcular el recargo.

Escribir el recargo ajustado a la derecha.

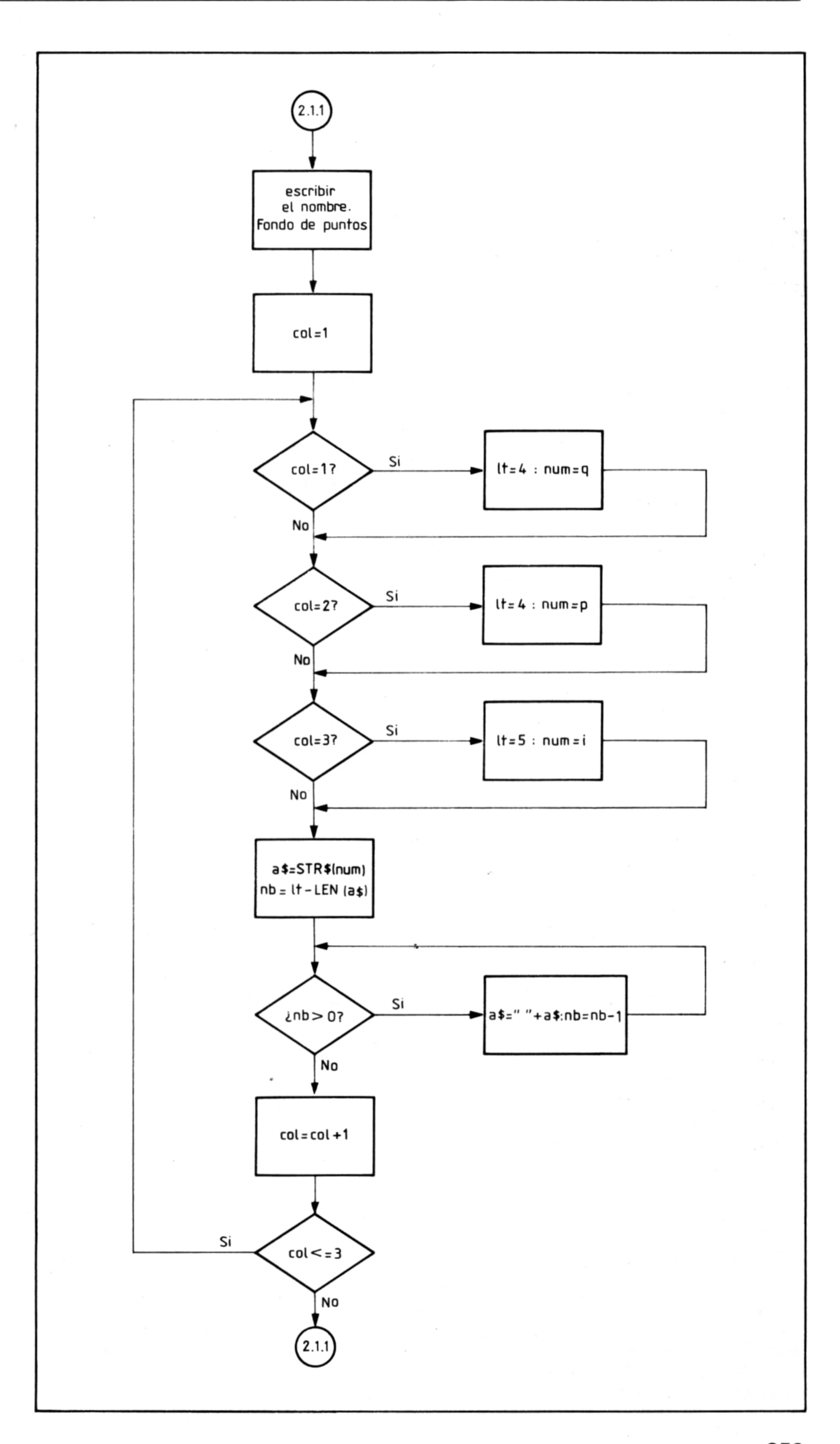


Figura 5: Ordinograma de desarrollo del bloque «Formateado y escritura».

Imprimir una raya.

Acumular el impuesto en el total.

Escribir el nuevo total ajustado por la derecha.

Escribir una doble raya de cierre.

= = = =Proceso 3

Las líneas de Proceso 3 indica que la lista corresponde al desarrollo del bloque 3 del ordinograma de primer nivel. Esta línea hace el mismo papel que los conectores de página, tal como hemos explicado en los ordinogramas anteriores.

Como se habrá dado cuenta no utilizamos ordinogramas muy largos, más o menos una página de papel tamaño folio. No es que se trate de una casualidad, o que el problema sea demasiado sencillo; es un efecto buscado en el diseño jerárquico. Cada unidad debe caber en una hoja de papel, de esta manera podemos comprender mucho mejor lo que hace el programa; un plano con gran cantidad de bloques de procesos y de flechas que se cruzan y van y vuelven no suelen clarificar ningún aspecto, antes bien contribuyen a complicar lo que dispuesto jerárquicamente es fácil de entender.

También se ha de señalar que cuando el bloque es lineal, es decir, sólo contiene bloque de procesos, una simple lista de acciones puede ser útil y no es necesario dibujar el proceso, tal como se ha hecho con el bloque de escritura de totales.

Para finalizar, en la figura 6 tiene la lista de variables según se ha descrito en la lección del BASIC estándar, para completar la información del programa.

En este ejemplo no se ha utilizado ninguna tabla de decisión pues no hay que tomar ninguna decisión excesivamente complicada.

VARIABLE	Descripción	Observaciones
f$	Contiene la línea de la fecha	Lon. menor de 22
d$	Contiene la línea de dirección	Lon. menor de 17
t	Total de la factura	
n$	Nombre del artículo	
q	Cantidad	
p	Precio unitario	
i	importe	
col	Indica la columna de impresión	1-3
lt	Longitud de la columna	4 o 5
nb	Blanco a añadir	
a$, b$	Variables auxiliares	
1, num	Variables auxiliares	
tpc	tanto por ciento de impuesto	0-100

Figura 6: Lista de variables del programa para la realización de la factura.

7.5 PRACTICA 2. LOS FILTROS LOGICOS

El segundo ejemplo práctico que vamos a realizar se trata del ordinograma del programa de filtros lógicos desarrollado en el capítulo anterior de prácticas con el microordenador.

En este ejemplo vamos a poner más énfasis en la documentación que en el diseño de un programa.

El programa completo se lo reproducimos a continuación.

```
  10 CLS
  20 PRINT AT 3,1;"MODELO NO:";
  30 LET a$="Motor Diesel 1)"
  40 IF LEN(a$)<23 THEN LET a$=" "+a$: GOTO 40
  50 PRINT AT 5,1;a$;
  60 LET a$="Traccion Delantera 2)"
  70 IF LEN(a$)<23 THEN LET a$=" "+a$: GOTO 70
  80 PRINT AT 6,1;a$;
  90 LET a$="Frenos de disco 3)"
 100 IF LEN(a$)<23 THEN LET a$=" "+a$:GOTO 100
 110 PRINT AT 7,1;a$;
 120 LET a$="5 Velocidades 4)"
 130 IF LEN(a$)<23 THEN LET a$=" "+a$:GOTO 130
 140 PRINT AT 8,1;a$;
1000 REM Parte de la entrada de datos
1010 LET d$="": LET t$=""
1020 LET f$="": LET v$=""
1030 LET n=0
1040 LET n=n+1
1050 PRINT AT 3,16;STR$(n);
1500 REM Parte de entrada propiamente dicha
1510 LET p=1: LET m$=""
1520 IF p>4 THEN GOTO 1600
1530 INPUT "Atributo "+STR$(p)+":";a$
1540 IF a$<>"s" and a$<>"n" THEN GOTO 1530
1550 LET m$=m$+a$
1560 PRINT AT p+4,24;a$
1570 LET p=p+1: GOTO 1520
1600 LET d$=d$+m$(1 TO 1)
1610 LET t$=t$+m$(2 TO 2)
1620 LET f$=f$+m$(3 TO 3)
1630 LET v$=v$+m$(4 TO 4)
1910 PRINT AT 5,24;" ";AT 6,24;" ";
1920 PRINT AT 7,24;" ";AT 8,24;" ";
1920 PRINT at 7,24;" ";at 8,24;" ";
1930 IF a$<>"n" THEN GOTO 1040
1940 CLS: REM Finaliza la parte de entrada de
          datos
2000 INPUT "Que pregunta desea formular:";p$
```

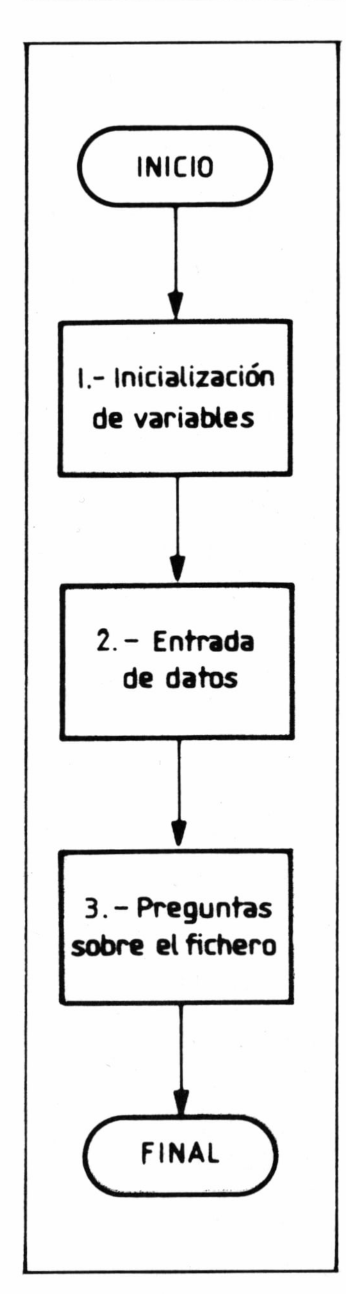


Figura 7: Ordinograma de primer nivel para los filtros lógicos.

```
2010 IF p$="" THEN GOTO 3000
2500 CLS : PRINT p$ : LET p=0
2510 LET p = p+1: IF p>n THEN GOTO 2900
2520 LET d = d$(p TO p) = "s"
2530 LET t = t$(p TO p) = "s"
2540 LET f = f$(p TO p) = "s"
2550 LET v = v$(p to p) = "s"
2560 IF NOT VAL p$ THEN GOTO 2510
2570 PRINT p;" "; : GOTO 2510
2900 GOTO 2000
3000 CLS: REM Ha finalizado el programa
```

Recordemos en primer lugar un poco cómo funcionaba el programa. La figura 7 tiene el ordinograma de primer nivel.

Se trata precisamente de ir recordando el programa a medida que vamos viendo el ordinograma.

El programa constaba de 3 bloques, en el primero se construía la pantalla que nos servía de fondo para la entrada de datos. Vea que en el bloque 1, comienza todo el proceso, según está explicado en el ordinograma.

En la segunda etapa o bloque se procedía a entrar los atributos de una serie de modelos de automóvil, referentes a unas características determinadas.

Finalmente, en la tercera etapa, y con el fichero entrado en la segunda etapa, se realizaban preguntas que nos permitían saber cuáles de los modelos cumplían unas características determinadas.

Una vez descrito el primer ordinograma vamos a desarrollar la primera etapa, para ello observe la figura 8. En esta etapa, lo primero a ejecutar es dejar la pantalla limpia; después hay que colocar los rótulos de los distintos atributos.

En el ordinograma el bloque de colocar rótulos está numerado como 1.1; en realidad, este bloque está desarrollado en la misma figura para un rótulo cualquiera. En el programa aparecerá tantas veces como rótulos debemos colocar.

Las operaciones a realizar en cada rótulo se especifican en el ordinograma 1.1. En primer lugar hay que dar a la variable a$ el valor del rótulo que queremos colocar, en el ordinograma se coloca el valor «Es un rótulo» para designar que sirve para cualquiera.

Una vez asignado a a$, hay que alargarlo con blancos por la derecha hasta que tenga la longitud deseada, en nuestro caso es 23.

Completada esta fase de alargamiento se imprime el rótulo en el lugar de la pantalla que le corresponde; los detalles están en el propio programa.

Una vez realizada la inicialización de las variables vamos a considerar la entrada de datos o bloque 2 del ordinograma inicial.

La figura 9 muestra este ordinograma. Una simple ojeada al mismo ya indica que el proceso es más complicado.

En primer lugar hay una fase de inicialización de las cadenas que van a contener el fichero y el contador de modelos.

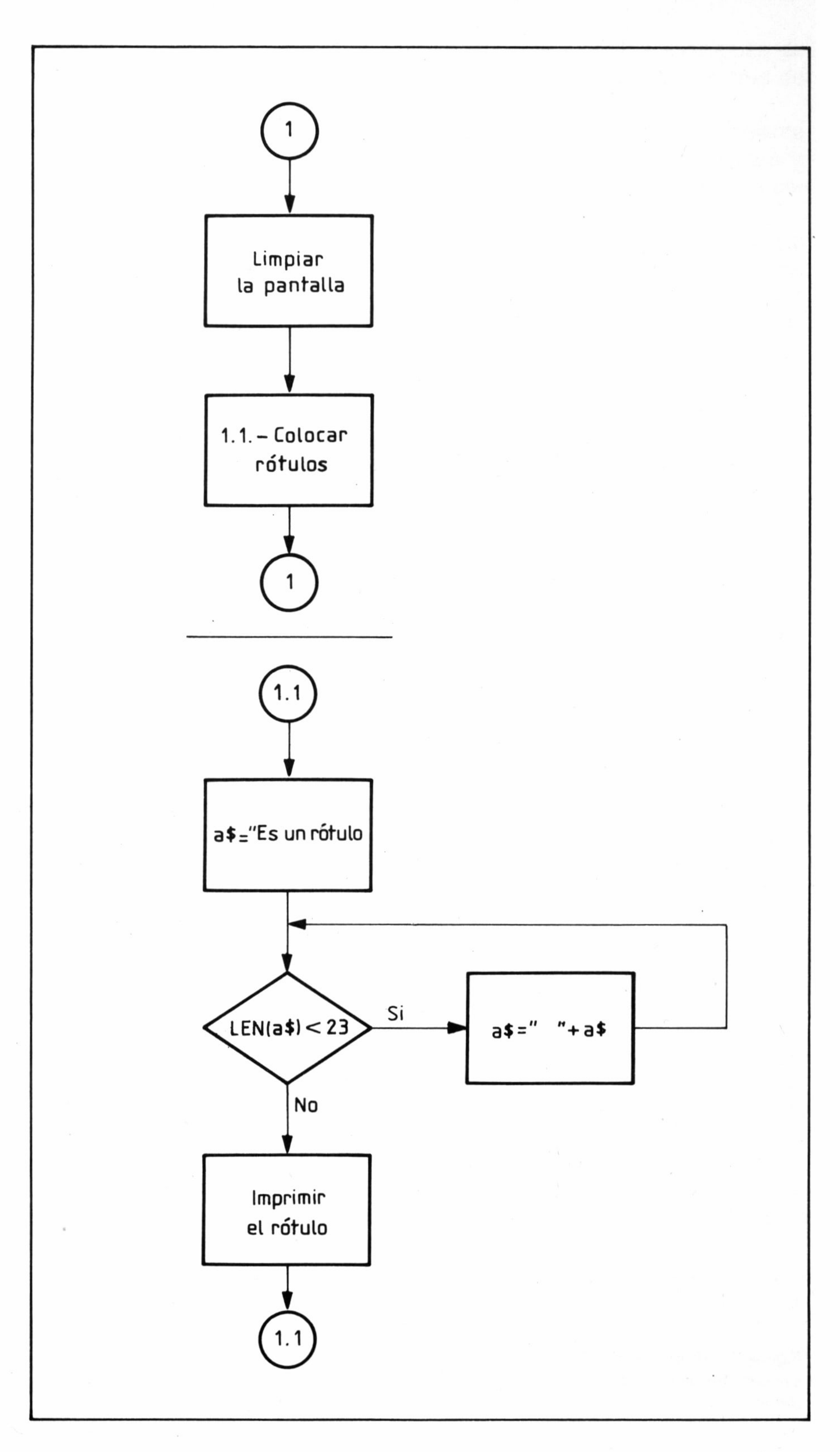


Figura 8: Ordinograma de desarrollo del bloque «Inicialización de variables».

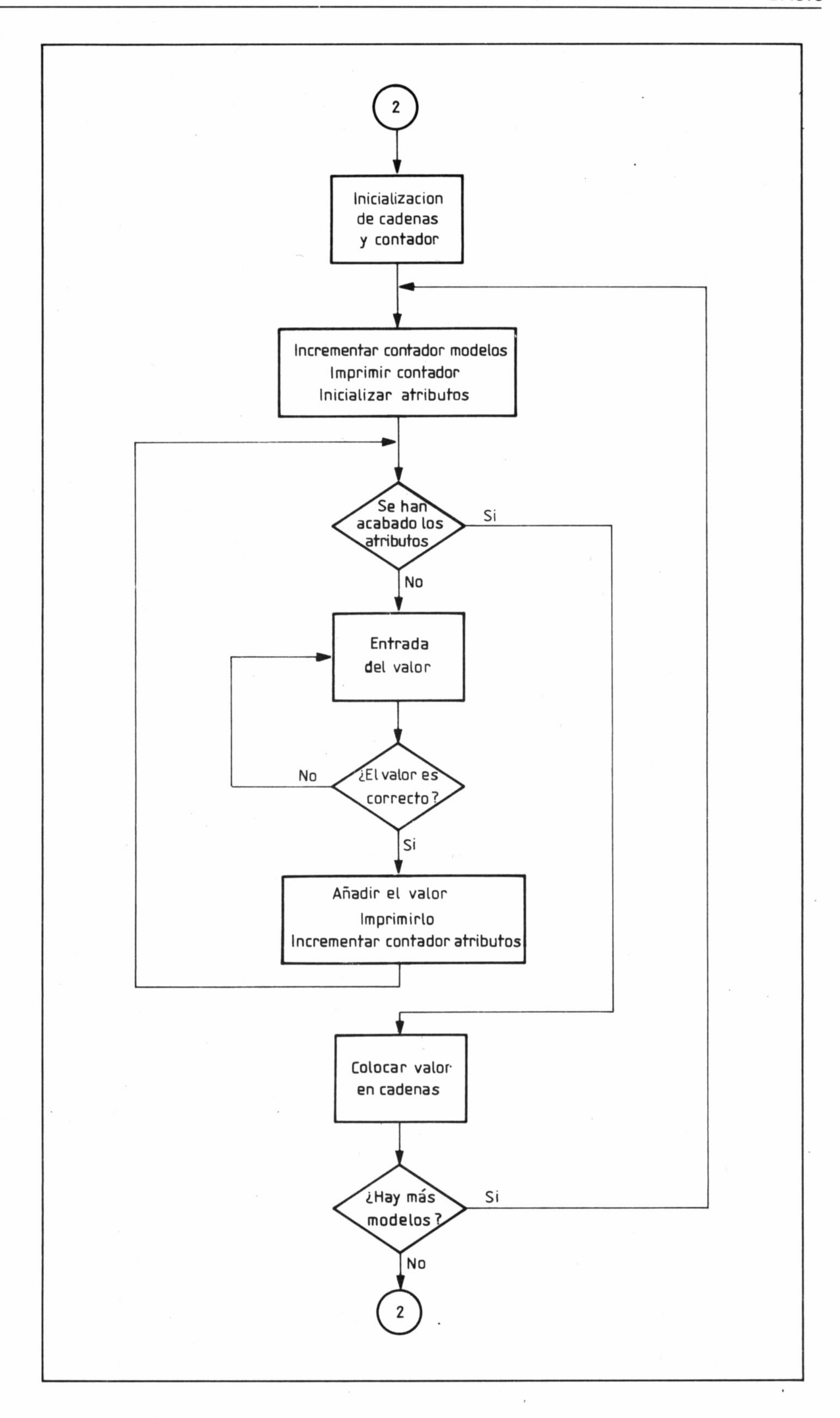


Figura 9: Ordinograma de desarrollo del bloque «Entrada de datos».

A continuación se incrementa el contador de modelos y después de colocarlo en pantalla se inicializa la cadena que contendrá los valores de los atributos.

Se pregunta a continuación, simbolizado en el ordinograma mediante el bloque de decisión, si ya se han acabado los atributos; si no los hemos acabado, se entra un valor para el atributo, mientras este valor no sea correcto se insiste en preguntar por un valor correcto.

En el momento que el valor es correcto entonces se añade el valor a los atributos, se imprime en la pantalla y se incrementa el contador que me indica qué atributo estoy entrando. El proceso en este momento se recicla hasta la pregunta de si hay más atributos. El proceso se repite hasta que se han entrado todos los atributos.

Cuando la respuesta a la pregunta de si se han acabado todos los atributos es afirmativa se va el bloque de proceso de colocar el valor en las cadenas del fichero.

Una vez realizada esta operación, el ordinograma nos muestra un bloque de decisión para saber si hemos finalizado con la entrada de modelos.

Si no se ha acabado, al proceso se recicla hasta incrementar el contador de modelos y preparar la entrada de los atributos para este modelo.

Este proceso se repite hasta que se terminan los modelos con lo que se finaliza el ordinograma del bloque 2.

Después de haber desarrollado el bloque 2 hay que pasar a describir el bloque 3, que es la parte del programa que nos permite seleccionar ciertos modelos de los entrados según las características formuladas en la pregunta. La figura 10 muestra el ordinograma de esta última fase.

Se inicia limpiando la pantalla de la entrada de datos.

Se realiza el proceso de entrada de la pregunta. Si la pregunta es distinta a la cadena vacía se pasa a la fase de cálculo.

En primer lugar, se limpia la pantalla y se imprime la pregunta formulada para que el usuario del programa pueda recordarla.

Se coloca el contador de modelos igual a cero; un bloque decisión nos indica si estamos en el último modelo.

En el caso de que tengamos aún modelos por considerar se calculan las variables lógicas asociadas al modelo y se evalúa la expresión que hemos realizado en la pregunta.

Un bloque de decisión nos recicla a otro modelo si no cumple. En caso contrario nos imprime el número de modelo que cumple con las características.

Cuando se han finalizado todos los modelos el bloque de decisión «Hay más modelos» nos recicla a una nueva pregunta.

El proceso continúa hasta que la pregunta que formulamos es vacía.

Habrá observado que el estilo de este ordinograma es diferente al realizado en la práctica primera, aquí no utilizamos el lenguaje BASIC para escribir dentro de un bloque de proceso.

Esto debería ser el ideal. Si escribimos en BASIC dentro de los bloques de proceso, el ordinograma pierde su generalidad; no se traducirá con facilidad a otro lenguaje.

Sin embargo, caemos en el riesgo de ser ambiguos; es probable que si utilizamos el lenguaje corriente se nos escapen detalles importantes, sobre todo en el control del valor de las variables.

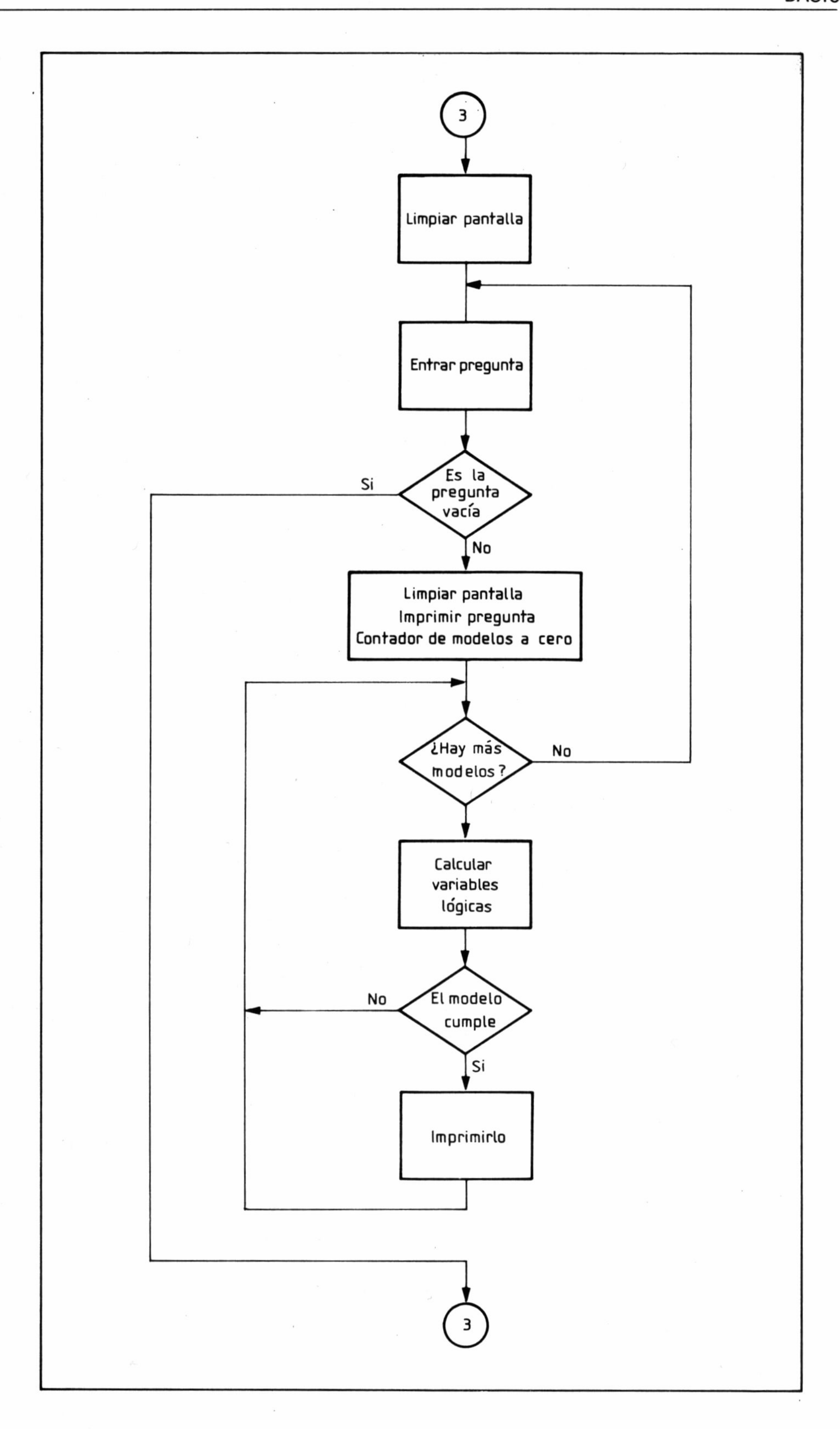


Figura 10: Ordinograma de desarrollo del bloque «Preguntas sobre el fichero».

Es muy difícil de precisar cual es la justa medida; diremos que este último estilo de ordinograma es más bien de documentación y, cuando vayamos a un diseño, es preferible utilizar en los bloques variables y expresiones escritas en algún lenguaje de programación.

Tenga en cuenta que, en lo que se refiere a los procesos, los lenguajes de programación son muy parecidos.

Capítulo 8

ESQUEMA DE CONTENIDO

8.1 ALMACENAMIENTO EN CINTA

Si usted dispone de una grabadora, ya conoce cómo manejarla para efectuar grabaciones o reproducir sonido.

Vamos a ver ahora la forma concreta de utilizarla como dispositivo de memoria auxiliar para nuestro ZX-SPECTRUM.

8.1.1 Conexión de la grabadora

Vamos a ver en primer lugar cómo conectar la grabadora con el ordenador. En el equipo del ZX-SPECTRUM se incluye un cable especial para efectuar esta conexión. Este cable tiene dos enchufes en cada extremo, de distinto color.

En la parte posterior de su ordenador, existen varios puntos de conexión. En este momento nos interesan los dos que están juntos, marcados como MIC (micrófono) y EAR (altavoz, EAR significa oído en inglés).

En cada uno de ellos introduciremos un enchufe, evidentemente de distinto color (por ejemplo el negro en EAR y el de color en MIC).

Ahora debemos conectar el enchufe del otro extremo del cable en el punto de conexión EAR de su grabadora, teniendo en cuenta que el color del enchufe debe ser el mismo que el conectado en EAR del ordenador, y a continuación el otro enchufe en el punto de conexión MIC de la grabadora.

Siguiendo con el ejemplo, el enchufe negro irá al punto EAR de la grabadora, y el de color a MIC, tal como se indica en la figura 1.

Con ello ya tendremos la conexión efectuada y podremos prepararnos para utilizar la grabadora.

Puede ser que su grabadora no posea el punto de conexión EAR. En este caso, habrá que conectar el enchufe en uno de los puntos de conexión para auriculares, si el aparato lo tiene, o bien al punto de conexión del altavoz externo.

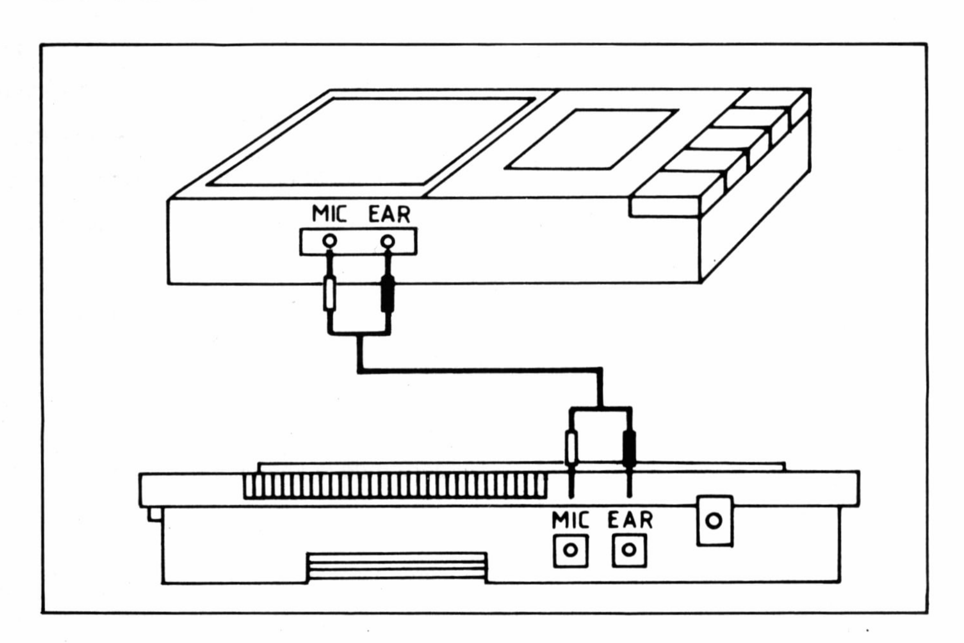


Figura 1 Conexiones de la cassette al ZX-SPECTRUM.

Por otra parte, si el enchufe del cable no encaja en el punto de conexión de la grabadora, necesitará un adaptador o un cable especial con los enchufes adecuados, que podrá obtener en una tienda de artículos eléctricos. Para conectar al ZX-SPECTRUM se requiere enchufes macho de 3,5 mm debiendo ser la señal de entrada a EAR de alrededor de 1 volt.

Ahora que ya va a comenzar a manejar la grabadora queremos hacerle una observación importante: procure siempre que al conectar los aparatos lo último en conectar a la corriente sea el ordenador y lo primero en desconectar también.

8.1.2. Explorando una cinta

Una vez conectada la grabadora con el ordenador, podemos empezar a trabajar. Con el equipo de su ordenador se le suministra una cinta. Colóquela en la grabadora situándola al principio. Tenga en cuenta el ajuste del volumen (y el tono si lo tiene) de su grabadora. Para el ZX-SPECTRUM, el tono debe ajustarse al valor máximo, y el volumen puede ir desde los dos tercios del máximo hasta el propio valor máximo. Usted mismo deberá irlo ajustando hasta encontrar la posición óptima.

El ajuste del volumen y del tono

Lo primero que haremos ahora será ver cuales son los programas que están grabados en esta cinta. Para ello, escribiremos el comando LOAD, pero seguido de un nombre extraño para asegurarnos de que la cinta no tiene ningún programa con este nombre. La palabra LOAD se obtiene pulsando la tecla de la J, en modo directo (K). Así, escribiremos por ejemplo:

```
LOAD "zzzz"
```

Una vez hayamos pulsado la tecla ENTER observaremos que la pantalla queda en blanco. Pondremos ahora la grabadora en marcha, el borde de la pantalla cambiará de color, volviéndose rojo o azul o una mezcla de ambos. Esto indica que el ordenador está buscando el programa. Instantes después, aparecen en el borde de la pantalla unas franjas rojas y azules que se van desplazando. Esto indica que el ordenador ha comenzado a recibir una señal. En un momento dado el fondo de pantalla cambia de amarillo y azul, mientras el ordenador lee el nombre del programa. Aparece entonces en la pantalla la palabra

Program:

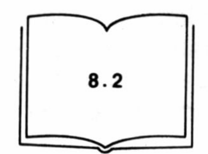
El ordenador indica qué programas ha encontrado en la cinta

seguida del nombre del programa que empieza en este punto de la cinta. Vuelven a aparecer las franjas azules y rojas, y finalmente empieza a cargar el programa, mientras que en el fondo de pantalla vuelven a aparecer las franjas azules y amarillas.

El ordenador va colocando todos los programas por los que pasa en la memoria, pero si no es el buscado continúa cargando el programa siguiente.

Dado que hemos escrito el nombre de un programa que no está en la cinta, el ordenador irá recorriéndola hasta el final buscando el programa

que le hemos indicado, mientras se irá repitiendo el proceso que hemos descrito. De esta manera irán apareciendo en pantalla y de forma sucesiva los nombres de los programas que han sido grabados en la cinta. Esto nos permitirá, pues, saber todos los programas que hay, y el orden en que se encuentran. Si además nuestra grabadora dispone de contador de vueltas, podremos saber el punto exacto de la cinta en el que empieza cada uno.



8.1.3 Como cargar los programas

Ahora podremos ya cargar el programa que deseemos. Para ello bastará colocar la cinta en la posición adecuada, y escribir la palabra LOAD seguida del nombre del programa.

Dado que ahora buscamos un programa que se encuentre en la cinta, cuando el ordenador lo encuentre, procederá a cargarlo —el proceso será el que ya conocemos—, sólo que en este caso aparecerá el nombre del programa en pantalla, y una vez finalizada la carga, la pantalla se pondrá en blanco y aparecerá el mensaje:

0 OK, 0:1

En este momento podremos parar la grabadora, y si lo deseamos, ejecutar el programa que acabamos de cargar.

Errores en la carga de un programa

Si al cargar un programa se produce un error, en la pantalla aparece el mensaje:

R Tape loading error

El origen del error puede variar: conexiones mal hechas, problemas con el tono y el volumen, etc. En este caso, se revisarán todos los pasos realizados.

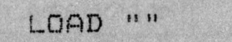
— Conexiones correctas

— Ajuste de tono y volumen

— Cinta mal colocada

A continuación se rebobinará la cinta y se escribirá de nuevo el comando LOAD seguido del nombre del programa para intentar de nuevo la carga.

El ZX-SPECTRUM presenta además una posibilidad adicional. Escribiendo:



el ordenador cargará el primer programa que encuentre en la cinta. Entre las comillas NO debe escribirse nada; ponga atención a no escribir un espacio en blanco.

Puede suceder que al cargar un programa aparezca en la pantalla el mensaje:

4 Out of memory

El programa es excesivamente largo

Este mensaje indica que se ha agotado la memoria. Esto puede haberse producido por un error en la carga del programa, en cuyo caso se deberá repetir el proceso de carga, o bien, si el programa es largo, porque en efecto éste no quepa en la memoria disponible del ordenador. En este caso se debe probablemente a que el ordenador que ha generado el programa disponía de más memoria. Mediante la instrucción CLEAR, que veremos más adelante en esta lección, se dispone de una pequeña capacidad de maniobra; pero, en general, este mensaje demuestra que no es posible cargar el programa en nuestro ordenador.

Ahora proceda a cargar el programa de la cinta de demostración que ha recibido con el ordenador. Puede pulsar LOAD, seguido de dobles comillas.

Encontrará en la lista de cosas que salen al explorar una cinta de demostración unas entidades que el ordenador denomina «Bytes:», en lugar de «Program:». Esto significa que en aquella zona de cinta hay grabados datos en lugar de programas. En el capítulo 13 veremos cómo se realiza este tipo de grabación y carga.



8.1.4 Cómo grabar los programas

Para efectuar la grabación de un programa con el ZX-SPECTRUM, debe tenerse en cuenta, en primer lugar, que se debe realizar una modificación en las conexiones, que consiste en desconectar el enchufe EAR, tal como se muestra en la figura 2.

Evidentemente, para grabar un programa deberemos haberlo escrito previamente. Supongamos que tenemos un programa para calcular la media aritmética de una lista de números; la longitud de la lista puede variar.

Para calcular la media aritmética de una lista de números, se suman los valores de todos los números y se divide por el número de elementos de la lista. Así, la media aritmética de tres números, por ejemplo 3, 5 y 7 se calcula:

$$\text{Media A.} = \frac{3 + 5 + 7}{3} = 5$$

El listado del programa es el siguiente:

```
10 REM ================================
20 REM Calculo de la media aritmetica
30 REM --------------------------------
40   CLS : LET s=0 : LET i=0
50   INPUT "Numero: ";n
60   LET i=i+1
70   LET s=s+n
```

```
80 INPUT "Otro numero (S/N): ";r$
90 IF r$="s" OR r$="S" THEN GOTO 50
100 PRINT "El valor medio de la lista es:";s/i
```

Para grabar un programa desconecte la clavija EAR

Vamos ahora a grabar este programa. En primer lugar deberemos cerciorarnos de que las conexiones estén situadas correctamente para efectuar la grabación: solamente el conector MIC del ZX-SPECTRUM está conectado con el conector MIC (o el equivalente) de su grabadora. Por lo tanto, desconecte el cable EAR si no lo ha hecho ya.

Escribiremos ahora la orden de grabación. Dado que el programa nos calcula el valor de la media aritmética le asignaremos un nombre que haga referencia a esta operación. La longitud máxima del nombre de un programa en el ZX-SPECTRUM es de diez caracteres. Por otra parte, dado

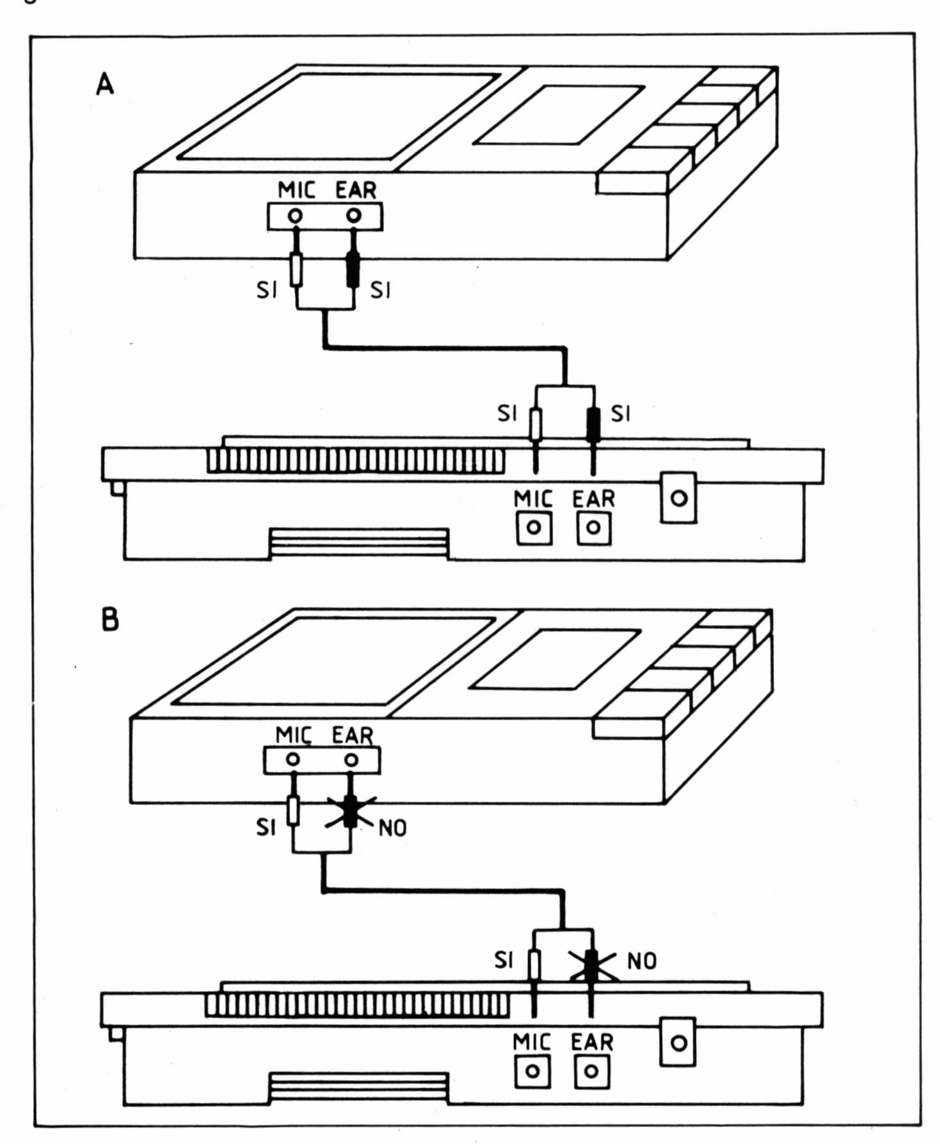


Figura 2 Situación de las conexiones del ZX-SPECTRUM. (A) para cargar. (B) para grabar.

que es la primera vez que grabamos este programa, escribiremos un 1 al final del nombre, indicando que es la versión 1. Si más adelante, hiciéramos un programa distinto para calcular la media, podríamos darle el mismo nombre pero variando el número de la versión (2,3,...). Así, el nombre de este programa será:

«media ari1»

El nombre que se da a un programa debe recordarse exactamente para poderlo recuperar

Por otra parte, debe tener en cuenta que el ZX-SPECTRUM distingue entre los nombres escritos en mayúsculas y los escritos en minúsculas. Así, si hemos escrito el nombre del programa en letras minúsculas, tal como acabamos de indicar, en el momento de leerlo deberemos escribirlo exactamente de la misma forma; el ordenador no lo identificaría con este programa si lo buscáramos escribiendo:

```
"MEDIA ARI1"
```

o bien como

```
"Media Ari1"
```

etc.

En este sentido es conveniente, cuando se graba un programa en una cinta, escribir en una lista junto a la cinta, la posición y el nombre exacto con que se ha grabado el programa, para facilitar la posterior carga del mismo.

Así pues, para grabar el programa colocaremos la cinta en la posición adecuada y escribiremos:

```
SAVE "media ari1"
```

La palabra SAVE se encuentra sobre la tecla de la letra S, con el ordenador en modo directo (K). Después de pulsar la tecla ENTER, desaparecerá de la pantalla la línea conteniendo la orden de grabación, y en su lugar aparecerá el mensaje:

Start tape, then press any key.

que significa:

Ponga en marcha la grabadora,
a continuación pulse cualquier tecla.

Si no teníamos preparada la grabadora, la podemos preparar ahora (hay que apretar las teclas REC y PLAY). Una vez esté todo dispuesto, para

indicar que la grabación ya puede comenzar, bastará pulsar una tecla cualquiera del ordenador, tal como indica el mensaje.

En este momento, se empieza a grabar el programa. En el fondo de la pantalla aparecen unas bandas rojas y azules que se desplazan lentamente, cambiando en un momento dado, en que aparecen bandas más estrechas y de color azul y amarillo. Esto indica que se está grabando el nombre del programa. Aparecen ahora de nuevo bandas azules y rojas, que vuelven a cambiar a azules y amarillas; es el momento que se inicia la grabación de programa, y se mantienen mientras dura la grabación del mismo. El tiempo de grabación depende naturalmente de la longitud del programa; si es largo puede durar varios segundos. Cuando finaliza la grabación, el ordenador da un mensaje, en la pantalla aparece:

0 OK, 0:1

En este momento podremos ya detener la cinta, y pasar a verificar el programa.

8.1.5 Verificación de un programa

Para tener la seguridad de que el programa ha sido correctamente grabado, habrá que verificarlo. El proceso para verificar un programa con el ZX-SPECTRUM, debe seguir los siguientes pasos:

— Rebobinar la cinta hasta el punto donde se ha iniciado la grabación

— Conectar el enchufe EAR del ZX-SPECTRUM con el enchufe EAR de la grabadora. No hace falta desconectar los enchufes del punto de conexión MIC (Figura 2)

— Dar la orden de verificación, que consiste en escribir la palabra VERIFY seguida del nombre del programa entre comillas. Una vez escrito, se pulsará la tecla ENTER y se pondrá la grabadora en marcha.

Así, escribiremos:

```
VERIFY "media aril"
```

La palabra VERIFY se encuentra sobre la tecla R, y se obtiene pulsando esta tecla simultáneamente con la de SYMBOL SHIFT, situando el ordenador en modo extendido (E).

La verificación tiene un comportamiento visual en pantalla semejante al LOAD

Una vez realizadas todas estas operaciones, se inicia el proceso de verificación. En pantalla se produce el mismo efecto que cuando se está cargando un programa con el comando LOAD; aparecen en primer lugar bandas rojas y azules, que cambian a azules y amarillas en el momento de leer el nombre del programa —éste aparece en pantalla—, vuelven a rojas y azules, pasando por último de nuevo a amarillas y azules en el momento de leer el programa. En este sentido, el proceso de verificación tiene un efecto igual al de carga en cuanto a la lectura de la cinta. Sin embargo,

difieren en un punto fundamental, el comando LOAD produce el borrado del contenido de la memoria del ordenador, previamente a la carga del programa. Evidentemente, el comando VERIFY no borra la memoria del ordenador, lo que hace es ir leyendo lo que se ha grabado y comparándolo con el contenido de la memoria. Si la verificación da el resultado correcto, en pantalla aparece el mensaje:

0 OK, 0:1

lo que indica que el contenido de la memoria coincide con lo que se ha registrado en la cinta.

Puede suceder que no aparezca este mensaje. Esto puede indicar que la grabación no había sido correcta, pero puede ser también que el problema se haya producido en la verificación. Deberá entonces en primer lugar repetir este procedimiento, repasando todos los pasos efectuados. Si aún así no obtiene el resultado esperado, deberá repetir el proceso de grabación, cerciorándose de efectuar correctamente todos los pasos que se precisan para ello.

Una vez ha verificado el programa puede dar como correcto el contenido de la cinta. Para comprobarlo, ponga un NEW en el ordenador y recargue el programa desde la cinta según lo que se le ha indicado en la instrucción LOAD.

Realmente puede observar que aún tiene problemas para cargar el programa, pero con toda seguridad provienen de la carga en condiciones no adecuadas y no de que el programa no esté en la cinta.

8.5

8.1.6 Combinar programas

Si nos interesa mezclar o fundir dos programas con el ZX-SPECTRUM, utilizaremos la orden MERGE.

El efecto de la misma, tal como ya hemos visto, es añadir a un programa que ya tenemos en la memoria del ordenador, otro programa que se va a cargar de la cinta, para ello bastará escribir la palabra MERGE seguida del nombre del programa que se desea cargar. El nombre del programa deberá tener también diez caracteres como máximo. El efecto de ejecutar esta instrucción en cuanto a la manipulación de la grabadora, fondo de pantalla, etc., es el mismo que cuando ejecutamos el comando LOAD.

Veámoslo con un ejemplo. Supongamos que queremos mezclar el programa que hemos escrito y grabado anteriormente, llamado «media ari1», con un programa que nos calcule la media geométrica.

Para calcular la media geométrica de una lista de números, se efectúa el producto de todos ellos, y se saca la raíz *n* de este producto, siendo *n* el número de elementos de la lista. Así, para obtener la media geométrica de 3, 5 y 7, haríamos:

$$\text{Media G.} = \sqrt{3 \times 5 \times 7} = 4.7$$

Obtener la raíz *n* de un número es equivalente a elevar este número al inverso de *n*. Vamos a comprobarlo con la raíz 2 (raíz cuadrada) del número 4. El ordenador dispone de una función que nos la calcula directa-

mente, pero podemos también determinarla calculando el valor del número 4 elevado a 1/2. Así, si hacemos:

```
PRINT SQR(4)
PRINT 4^(1/2)
```

obtenemos el mismo resultado en ambos casos. Podemos por tanto calcular la raíz *n* de cualquier número aunque el ordenador no tenga la función directa, elevando este número al inverso de *n*.

Escribiremos ahora el programa en el ordenador, y una vez finalizado y comprobado, cargaremos el de calcular la media aritmética. Veamos el listado del primero:

```
200 REM Calculo de la media geometrica
210   CLS : LET p=1 : LET i=0
220   INPUT "Numero: ";n
230   LET i=i+1
240   LET p=p*n
250   INPUT "Otro numero (S/N): ";r$
260   IF r$="s" OR r$="S" THEN GOTO 220
270   PRINT "La media geometrica es ";p^(1/i)
```

Una vez comprobado su funcionamiento, cargaremos el programa de la media aritmética. Para ello situaremos la cinta en la posición adecuada, efectuaremos las conexiones de MIC y EAR, y escribiremos:

MERGE «media ari1»

La palabra MERGE se encuentra sobre la tecla de la letra T, y se obtiene pulsando ésta simultáneamente con la tecla SYMBOL SHIFT, en modo extendido (E).

Pulsaremos ENTER y pondremos la grabadora en marcha. Veremos aparecer la conocida secuencia de bandas rojas y azules, después azules y amarillas, mientras se efectúa la lectura del nombre, y aparece éste en pantalla.

Program: media ari1

de nuevo azules y rojas y finalmente azules y amarillas mientras se hace efectiva la carga del programa.

Si todo va bien, cuando llegue al término del mismo, aparecerá el mensaje

0 OK, 0:1

Si efectuamos ahora el listado, escribiendo LIST, veremos el programa resultante de mezclar los dos anteriores:

```
 10 REM ====================================
 20 REM Calculo de la media aritmetica
 30 REM ------------------------------------
 40   CLS : LET s=0 : LET i=0
 50   INPUT "Numero: ";n
 60   LET i=i+1
 70   LET s=s+n
 80   INPUT "Otro numero (S/N): ";r$
 90   IF r$="s" OR r$="S" THEN GOTO 50
100   PRINT "El valor medio de la lista es: ";s/i
200 REM Calculo de la media geometrica
210   CLS : LET p=1 : LET i=0
220   INPUT "Numero: ";n
230   LET i=i+1
240   LET p=p*n
250   INPUT "Otro numero (S/N): ";r$
260   IF r$="s" OR r$="S" THEN GOTO 220
270   PRINT "La media geometrica es ";p^(1/i)
```

Ahora modificaremos el programa de forma que podamos especificar cual de las dos medias deseamos calcular. Así, incluiremos las líneas:

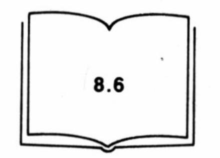
```
  1   CLS
  2   INPUT "Desea Media Arit.(a), Geom.(g), Fin(f): ";r$
  4   IF r$="f" THEN GOTO 9999
  6   IF r$="g" THEN GOTO 200
  8   IF r$="a" THEN GOTO 40
  9   GOTO 2
110   GOTO 2
280   GOTO 2
```

El programa completo quedará ahora:

```
  1   CLS
  2   INPUT "Desea Media Arit.(a), Geom.(g), Fin(f): ";r$
  4   IF r$="f" THEN GOTO 9999
  6   IF r$="g" THEN GOTO 200
  8   IF r$="a" THEN GOTO 40
  9   GOTO 2
 10 REM ================================
 20 REM Calculo de la media aritmetica
 30 REM --------------------------------
 40   CLS : LET s=0 : LET i=0
 50   INPUT "Numero: ";n
 60   LET i=i+1
 70   LET s=s+n
 80   INPUT "Otro numero (S/N): ";r$
 90   IF r$="s" OR r$="S" THEN GOTO 50
100   PRINT "El valor medio de la lista es: ";s/i
110   GOTO 2
200 REM Calculo de la media geometrica
210   CLS : LET p=1 : LET i=0
220   INPUT "Numero: ";n
```

```
230   LET i=i+1
240   LET p=p*n
250   INPUT "Otro numero (S/N): ";r$
260   IF r$="s" OR r$="S" THEN GOTO 220
270   PRINT "La media geometrica es ";p^(1/i)
280   GOTO 2
```

De esta forma pues, hemos conseguido mezclar los dos programas, tal como deseábamos, con una ventaja adicional, y es que hemos podido probarlos por separado, asegurando que cada uno funciona. Esto es muy importante, dado que siempre es más fácil atacar los problemas que presenta un programa corto. De esta forma, para realizar un programa complejo, podremos dividirlo, resolviendo cada parte por separado, y cuando estemos seguros del funcionamiento de cada una, podremos construir el programa completo utilizando el MERGE.



8.1.7 Carga con ejecución inmediata

Si nos interesa que un programa se ponga en funcionamiento de forma inmediata al efectuar la carga del mismo, deberemos utilizar una forma particular de la orden de grabación. Esta forma en el ZX-SPECTRUM consistirá en escribir la orden SAVE seguida del nombre del programa, tal como hemos visto, pero seguida además de la palabra LINE y de un número o expresión numérica. Así, se escribirá:

```
SAVE "nombre" LINE expresion numerica
```

La palabra LINE se encuentra sobre la tecla correspondiente al número 3, y se obtiene pulsando esta tecla en modo extendido (E), simultáneamente con la de SYMBOL SHIFT.

El nombre del programa puede tener un máximo de 10 caracteres tal como hemos visto en la utilización del SAVE.

El resultado de la expresión numérica escrita después de la palabra LINE, indicará el punto de inicio del programa cuando lo carguemos. Así, el resultado de esta expresión será redondeado (si es necesario), y el valor obtenido indicará el número de línea en que empezará la ejecución. Puede suceder que:

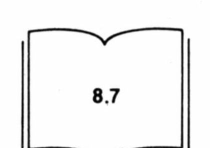
— El resultado sea el valor 1: el programa se empezará a ejecutar por la primera línea.

— El resultado sea un número de línea existente en el programa: éste se ejecutará a partir de este número de línea.

— El resultado sea un número de línea que no está en el programa: éste se empezará a ejecutar por la línea inmediatamente posterior a este valor.

Evidentemente lo que hemos dicho es válido si en lugar de una expresión, se ha escrito directamente un valor numérico después de la palabra LINE.

Si un programa se ha grabado utilizando el sistema de carga para ejecución inmediata, no podremos utilizar el comando MERGE para cargarlo posteriormente con el fin de mezclarlo con otro programa.



8.2 INSTRUCCION CLEAR

En la figura 3 se muestra el mapa de memoria del ZX-SPECTRUM, indicando las diferentes zonas en que ésta se encuentra dividida.

La parte izquierda de la figura representa a escala la memoria del ZX-SPECTRUM. Las 16 K primeras son ROM y las 16 restantes para el ZX-SPECTRUM de 16K o las 48 restantes para el ZX-SPECTRUM de 48K son RAM.

La distribución de la memoria según la utiliza el sistema es:

Distribución de la memoria en el ZX-SPECTRUM

1. En la ROM se sitúa el intérprete del lenguaje BASIC y los programas de servicio, tales como la copia de un programa en cassette, la gestión de la impresora, etc.

2. Las primeras 9K (aproximadamente) de la RAM se destinan a las comunicaciones con los periféricos como la zona de memorización de los símbolos que hay en la pantalla, el color de la tinta y el color del borde, o la memoria que se precisa para el tráfico de caracteres con la impresora, etc.

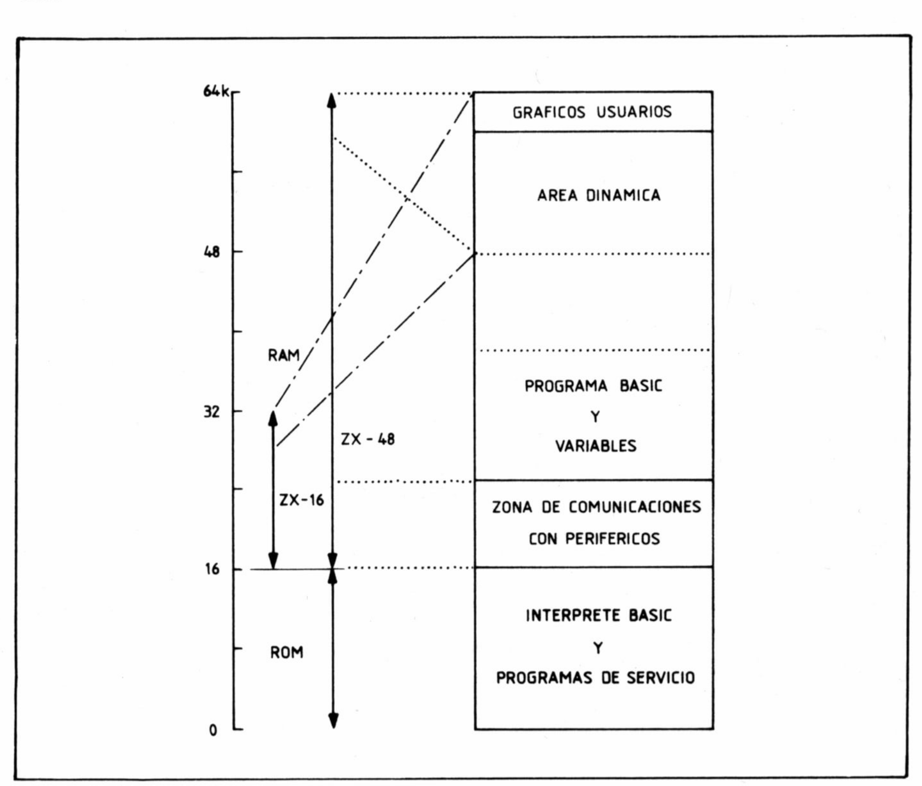


Figura 3. Mapa de memoria del ZX-SPECTRUM de 48 Kby y de 16 Kby.

Dése cuenta que es necesario que todo esto esté en la memoria RAM. Esencialmente son datos variables que necesita el sistema y, por lo tanto, deben estar situados en una memoria que pueda escribirse o alterarse.

3. A partir de este punto hacia arriba está la zona de memoria que guarda nuestro programa BASIC y las variables que estamos utilizando.

El límite de esta zona es impreciso pues depende de la longitud de nuestro programa y del número de variables que utilicemos.

4. Finalmente en la parte superior existe la zona de gráficos definidos por el usuario (se estudiarán en el capítulo 14 del tomo IV) y la denominada área dinámica.

¿Qué es el área dinámica?

Proporcionalmente ocupan poco espacio, la línea de puntos que conecta estas zonas a la escala nos indican la proporción correcta. Para el caso del ZX-SPECTRUM de 16K esta zona está situada al borde de las 32 K tal como indican las líneas de punto y raya. Es necesario hacer un comentario sobre el área dinámica. Se llama dinámica porque el límite inferior no está fijado, el valor que se da en la figura es aproximado. Durante la ejecución de un programa surgen muchas situaciones en las que hay que recordar cosas durante poco tiempo sin que estos valores sean variables de nuestro programa.

Consideremos el ejemplo de realizar el cálculo de la expresión (3+4)*(5+6).

La máquina tiene que sumar 3 + 4 y obtener 7, pero antes de proceder a la multiplicación debe realizar la suma 5 + 6, pero sin olvidar el 7 obtenido.

En este momento entra en juego el área dinámica. En ella se almacena el 7. Para poder almacenar este 7, es necesario que el límite inferior se haya corrido lo suficiente para recordar este valor.

La máquina procede a continuación a efectuar la suma de 5 + 6, obteniendo 11 que ahora debe multiplicar por el último valor guardado en el área dinámica, el 7 de la operación anterior. Obtiene el valor del área dinámica y como ya no lo va a necesitar más puede correr hacia arriba el límite inferior del área dinámica, reduciéndola otra vez a su valor inicial.

Finalmente realiza la operación de multiplicar 7 * 11 y obtiene 77.

Situaciones como las descritas son muy frecuentes en el cálculo interno, mucho más de lo que uno puede pensar en primera instancia. Por lo tanto, este límite va cambiando a lo largo de la ejecución del programa, normalmente sin alcanzar valores demasiado grandes.

La instrucción CLEAR en el ZX-SPECTRUM produce la inicialización de la memoria. Esto supone el borrado de los valores asignados a todas las variables, incluyendo las tablas o listas que se hayan definido, así como el restablecimiento de todas las condiciones iniciales de la memoria del sistema, de la misma forma que cuando se escribe el comando RUN. Además de este efecto interno, se produce un efecto externo, que es el borrado de pantalla, idéntico al que produce la instrucción CLS.

La palabra CLEAR se encuentra sobre la tecla de la letra X, y se obtiene pulsando esta tecla en modo normal (K). Para obtener este efecto, escribiremos pues:

```
CLEAR
```

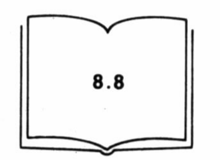
El efecto interno puede describirse como una reducción del área dinámica a cero y una reducción del área de nuestro programa a únicamente el texto del programa.

Contrariamente a lo que sucede en el BASIC estándard, las variables numéricas y las variables textuales no están separadas en dos zonas de memoria distinta.

La instrucción CLEAR seguida de un número

La instrucción CLEAR seguida de un número no se aplica del mismo modo que en el BASIC estándard. Por otra parte, al no estar las variables separadas no tenemos que preocuparnos si nuestro programa utiliza muchas o pocas variables numéricas o textuales.

La utilización del CLEAR seguido de un número en el ZX-SPECTRUM tiene un significado muy distinto y complejo de entender, por lo tanto, diferiremos su estudio hasta el capítulo 17, en el tomo V en que habla del lenguaje máquina.



8.3 UTILIZACION DE LA IMPRESORA

Si disponemos de impresora conectada al ZX-SPECTRUM, podremos utilizarla para obtener listados de nuestros programas, así como la impresión en papel de los resultados de los mismos.

8.3.1 Impresión de un listado

Para obtener impreso el listado de un programa, bastará escribir:

```
LLIST
```

La palabra LLIST se encuentra sobre la letra V, y se obtiene pulsando esta tecla en modo extendido.

Cuando se ejecuta esta instrucción se obtiene el listado del programa que se encuentra en memoria en este momento. Mientras dura la impresión del programa, no se altera la información visualizada en pantalla, pero, si se desea, se puede detener el proceso de escritura pulsando la tecla de interrupción.

Si únicamente deseamos el listado de una parte del programa, escribiremos la palabra LLIST seguida del número de la primera línea que se desea imprimir. De esta forma obtendremos el listado desde esta línea hasta el final. Si se escribe un número de línea que no se encuentra en el programa, el listado se inicia en la línea siguiente a este número. En lugar de un número se puede utilizar una expresión numérica. Si el resultado de la misma no es un número entero, su valor será redondeado, y se obtendrá el mismo efecto que en el caso de un número.

En el caso de ZX-SPECTRUM, al contrario de lo que ocurre en la mayoría de máquinas, la ejecución del comando no bloquea la máquina si no está conectada la impresora. Es más, actúa como si realmente se hubiera impreso.

8.3.2 Impresión de resultados

Para imprimir un resultado en papel con el ZX-SPECTRUM utilizaremos la instrucción LPRINT. Esta instrucción nos permitirá obtener impreso lo mismo que podemos obtener en pantalla con la instrucción PRINT. En este sentido es válido todo lo que se diga para PRINT; cambiará únicamente el soporte en que el ordenador nos presenta el resultado. La palabra LPRINT se encuentra sobre la C, y se obtiene pulsando esta tecla en modo extendido (E).

8.3.3 Copia de pantalla

El ZX-SPECTRUM permite efectuar una copia de la información visualizada en la pantalla. Para ello se escribirá:

```
COPY
```

La palabra COPY se encuentra sobre la letra Z, y se obtiene pulsando esta tecla en modo normal (K). La copia podrá interrumpirse utilizando la tecla de interrupción.

Si se desea copiar un listado, se deberá visualizar previamente en pantalla mediante la instrucción LIST. Sin embargo, si el listado ocupa más de una pantalla como ocurre frecuentemente, sólo obtendremos el trozo que se visualice en ella en el momento de ejecutar esta orden. De esta forma, para obtener impreso el listado completo, deberemos irlo visualizando en pantalla en forma parcial, e ir realizando sucesivamente la copia de cada parte.

Por otra parte, podrán obtenerse de la misma forma la impresión de resultados, dibujos o gráficos que se hayan obtenido en la pantalla, dado que el comando COPY traslada a papel todos los puntos que estén escritos o dibujados en la pantalla. Los colores de la tinta se imprimirán en el papel en negro; los colores del papel (fondo) no se imprimen.

8.4 PRACTICA. MEZCLA DE PROGRAMAS

La práctica 2 del capítulo 6, titulada la construcción de filtros, es un buen ejemplo para estudiar la operación de MERGE y las demás de manipulación de la grabadora.

El programa tiene dos partes bien diferenciadas, la entrada de datos y la selección de información.

La entrada de datos cubre las instrucciones 10 a la 2040 y la selección de información de la 2000 hasta la 3000.

El objetivo de la práctica consiste en introducir la primera parte en el ordenador y guardarlo en el cassette con el nombre «sled1»; siglas que quieren recordar a Selección Lógica: Entrada de Datos versión 1.

Introducir la segunda parte y guardarla en el cassette con el nombre «slsi1»; siglas construidas a partir de Selección Lógica: Selección de información versión 1.

Posteriormente mezclaremos los dos programas y los guardaremos en la cinta con el nombre «sl1».

Antes de realizar el trabajo realmente escriba una lista de operaciones a realizar, de la manera más detallada posible.

Le damos a continuación una realizada por nosotros. Compárela con la suya y analice las diferencias, si las hay; en cualquier caso nuestro método no es el único, pero la experiencia indica que funciona bien.

En último término realice realmente los dos planos y vea si el resultado final es correcto.

La lista de operaciones es la siguiente:

1. Conecte el ordenador y teclee las instrucciones del programa que aparecen en el apartado 6.4.2.2. del capítulo 6 de las Prácticas con el microordenador.

2. Prepare la grabadora con una cinta limpia que colocará al inicio de una de las caras mediante la tecla de rebobinar de la grabadora. Si tiene contador de vueltas póngalo a cero. Desconecte la clavija conectada a EAR en el ordenador.

3. Ejecute el comando SAVE «sled1». El ordenador se parará en el mensaje de que para «continuar debe apretar una tecla». No la apriete aún.

4. Apriete las teclas de REC y PLAY del cassette y la cinta se pondrá en marcha.

5. Deje que dé cinco o seis vueltas y entonces apriete una tecla del ordenador. Aparecerán en pantalla las franjas rojas y azules y posteriormente las amarillas y azules.

6. Cuando se haya acabado el proceso, el ordenador nos indicará el fin con el mensaje 0 OK, 0:1. Apriete entonces el STOP de la grabadora. No es necesario que se dé prisa, alguna vuelta más grabando no tiene mayor importancia que el perder un poco de cinta. Si dispone de cuentavueltas anote donde ha finalizado el programa.

7. Rebobine la cinta, conecte la clavija EAR al ordenador.

8. Ejecute el comando VERIFY.

9. Pulse la tecla PLAY de la grabadora. Aparecerán las franjas azules y rojas y luego las amarillas y azules.

10. Si el proceso de verificación es correcto aparecerá en pantalla el 0 OK, 0:1. En caso contrario tiene la opción de ir otra vez al paso 7, si cree que el problema proviene de la verificación, o bien al paso 2 si cree que el problema proviene del comando SAVE. No debe sobrepasar este punto hasta conseguir que la verificación sea correcta.

11. Coloque un NEW en el ordenador y teclee el programa del apartado 6.4.3.2. del capítulo 6 de prácticas.

12. Prepare la grabadora con la cinta anterior. Ahora debe colocarla más allá del programa grabado en la primera fase. Si dispone de contador

de vueltas ya sabe dónde se acaba el programa anterior y puede situarse mediante la tecla de avance rápido de su grabadora. Si no dispone de contador de vueltas, puede colocarse aproximadamente en la zona mediante la tecla de avance rápido y darle entonces a la tecla PLAY para poder oír cuándo acaba el programa anterior (incluso puede grabar con el micrófono de audio unas palabras suyas indicando que en aquel punto acaba el programa). Desconecte la clavija conectada a EAR en el ordenador.

13. Ejecute el comando SAVE «slsi1». El ordenador se parará en el mensaje de que para continuar debe apretar una tecla. No la apriete aún.

14. Apriete las teclas de REC y PLAY del cassette y la cinta se pondrá en marcha.

15. Deje que dé cinco o seis vueltas y entonces apriete una tecla del ordenador. Aparecerán en pantalla las franjas rojas y azules y posteriormente las amarillas y azules.

16. Cuando se haya acabado el proceso, el ordenador nos indicará el fin con el mensaje 0 OK, 0:1. Apriete entonces el STOP de la grabadora. No es necesario que se dé prisa. Si dispone de cuentavueltas anote dónde ha finalizado el programa.

17. Rebobine la cinta, hasta el inicio del programa, mediante el contador de vueltas o escuchando la cinta. Conecte la clavija EAR al ordenador.

18. Ejecute el comando VERIFY.

19. Pulse la tecla PLAY de la grabadora. Aparecerán las franjas azules y rojas y luego las amarillas y azules.

20. Si el proceso de verificación es correcto aparecerá en pantalla el 0 OK, 0:1. En caso contrario tiene la opción de ir otra vez al paso 17, si cree que el problema proviene de la verificación, o bien al paso 12 si cree que el problema proviene del comando SAVE. No debe sobrepasar este punto hasta conseguir que la verificación sea correcta.

21. Ahora podemos realizar el MERGE de los programas. Para ello coloque la cinta al inicio que es donde empieza el programa «sled1».

22. Escriba en el ordenador el comando MERGE «sled1» y la tecla fin de línea.

23. Apriete la tecla PLAY de la grabadora. Al cabo de unas vueltas empezarán a aparecer a franjas azules y rojas y después las franjas amarillas y azules. En la pantalla aparecerá «Program: sled1».

24. Cuando haya finalizado la carga aparecerá 0 OK, 0:1. Si no ocurre así, es que hay algún problema y debemos volver al paso 21.

25. En este momento disponemos en memoria de los programas mezclados. Deberemos corregir las sentencias 2000 y 2010 pues como eran comunes ha tomado las del programa «sled1», que son precisamente las que no nos interesan y copiar las del apartado 6.4.3.2. Y deberemos borrar las sentencias 2020, 2030 y 2040 porque no deben aparecer en el programa final.

26. Comprobamos el programa ejecutando las pruebas que nos sugiere el capítulo 6 de prácticas.

27. Prepare la grabadora con la cinta anterior. Ahora debe colocarla más allá del programa grabado en la segunda fase, es decir «slsi1». Si dispone de contador de vueltas ya sabe donde se acaba el programa anterior y puede situarse mediante la tecla de avance rápido de su grabadora. Si no dispone de contador de vueltas, puede colocarse aproximadamente en la zona mediante la tecla de avance rápido y darle entonces a la tecla PLAY para poder oír cuando acaba el programa anterior. Desconecte la clavija conectada a EAR en el ordenador.

28. Ejecute el comando SAVE «sl1». El ordenador se parará en el mensaje: «para continuar debe apretar una tecla». No la apriete aún.

29. Apriete las teclas de REC y PLAY del cassette y la cinta se pondrá en marcha.

30. Deje que dé cinco o seis vueltas y entonces apriete una tecla del ordenador. Aparecerán en pantalla las franjas rojas y azules y posteriormente las amarillas y azules.

31. Cuando se haya acabado el proceso, el ordenador nos indicará el fin con el mensaje 0 OK, 0:1. Apriete entonces el STOP de la grabadora. Si dispone de cuentavueltas anote dónde ha finalizado el programa.

32. Rebobine la cinta, hasta el inicio del programa, mediante el contador de vueltas o escuchando la cinta. Conecte la clavija EAR al ordenador.

33. Ejecute el comando VERIFY.

34. Pulse la tecla PLAY de la grabadora. Aparecerán las franjas azules y rojas y luego las amarillas y azules.

35. Si el proceso de verificación es correcto aparecerá en pantalla el 0 OK, 0:1. En caso contrario tiene la opción de ir otra vez al paso 32, si cree que el problema proviene de la verificación, o bien al paso 27 si cree que el problema proviene del comando SAVE. No debe sobrepasar este punto hasta conseguir que la verificación sea correcta.

Ahora dispone del programa del capítulo 6 de prácticas grabado en la cinta, en tres trozos. El primero contiene la fase de entrada de datos, el segundo contiene la fase de recuperación de información y el tercero el programa completo.

Es el momento de etiquetar la cinta del cassette para disponerla en el archivo de programas.

Para finalizar unos consejos para tener bien ordenado su archivo de cintas:

a) Rotulación: Escriba siempre sobre las cintas físicas, aunque sea a lápiz. La caja del cassette no está indisolublemente ligada a la cinta.

b) Duración: Utilice cintas de corta duración. Las búsquedas en cintas largas son precisamente largas. No se trata de música, el ordenador sólo puede escuchar un programa detrás de otro.

c) Espaciado: Deje espacio suficiente entre varios programas. El posicionamiento es difícil y, aunque Vd. disponga de cuenta vueltas en su

grabadora, puede tener la necesidad de cargar el programa en otra grabadora, ya sea por reparación ya sea porque se trata de otro ordenador.

d) Cinta de trabajo: Utilice una cinta distinta para el trabajo de poner a punto un programa. Contendrá las distintas versiones del programa. Cuando el programa es definitivo colóquelo en una cinta de acuerdo con la clasificación que se haya impuesto.

e) Copias de seguridad: Duplique las cintas de archivo, evitará tener que rehacer el programa si una de ellas sufre un accidente. Es bastante sencillo que la cinta se enrede en la grabadora y quede inutilizada.

Índice

Capítulo 7

Capítulo 8

ENCICLOPEDIA DEL BASIC

SPECTRUM

2

- Las instrucciones IF... THEN, END y STOP
- La instrucción IF con ELSE
- La instrucción PAUSE
- El caso de muchas alternativas
- Datos de tipo booleano: NOT, AND, OR y XOR
- Expresiones con operadores booleanos y con el ordenador
- La toma de decisiones
- Intervalos numéricos
- Decisiones con dos variables
- Propiedades de AND y OR mediante dos preguntas sucesivas
- Propiedad distributiva
- Leyes de Morgan
- Los operadores lógicos
- La construcción de filtros
- La función INKEY$
- Diseño de programas
- Ordinogramas
- Representación gráfica de las funciones
- Codificación
- Tablas de decisión y depuración de tablas
- Almacenamiento de programas: leer, grabar y verificar
- Mezcla y encadenado de programas
- Organización de la memoria
- La instrucción CLEAR
- La impresora: instrucciones LLIST y LPRINT

ceac